EDA技术与数字系统设计

包 明 赵明富 陈渝光 编著

北京航空航天大学出版社
http://www.buaapress.com.cn

责任编辑：刘晓明
封面设计：艺铭设计

EDA技术与数字系统设计

包明 赵明富 陈渝光 编著

EDA技术与数字系统设计

ISBN 7-81077-201-5

9 787810 772013 >

ISBN 7-81077-201-5/TP·112

定价：21.00元

EDA 技术与数字系统设计

包　明　赵明富　陈渝光　编著

北京航空航天大学出版社
http://www.buaapress.com.cn

内 容 简 介

本书较系统地介绍了 EDA 技术及现代数字系统的设计方法。全书分为三部分：虚拟电子工作台、大规模可编程逻辑器件和现代数字系统设计。主要包括：EDA 技术的基本概念、特征和工具；虚拟电子工作台的特点和电子电路分析方法，如交直流分析、瞬态分析、传递函数分析、零-极点分析、温度扫描分析等；大规模可编程逻辑器件的基本结构、资源和工作原理。重点介绍了 Altera 和 Lattice 公司的可编程逻辑器件、EDA 开发软件的设计方法。最后介绍了现代数字系统的基本结构、设计特点和自顶向下的设计方法以及描述数字系统的常用工具——算法流程图和 ASM 图，并且给出了利用 EDA 开发工具进行数字系统的设计实例。

本书可作为高等院校电类、机电类专业本专科生和研究生学习 EDA 技术及其应用的教材，也可供电子系统工程技术人员参考。

图书在版编目(CIP)数据

EDA 技术与数字系统设计/包明等编著. —北京：北京航空航天大学出版社，2002.7

ISBN 7-81077-201-5

Ⅰ.E… Ⅱ.包… Ⅲ.①电子电路 计算机辅助设计②数字系统—系统设计 Ⅳ.①TN702②TP271

中国版本图书馆 CIP 数据核字(2002)第 030907 号

EDA 技术与数字系统设计

包 明 赵明富 陈渝光 编著

责任编辑 刘晓明

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(100083) 发行部电话：(010)82317024 传真：(010)82328026

http://www.buaapress.com.cn

E-mail:pressell@publica.bj.cninfo.net

河北省涿州市新华印刷厂印装 各地书店经销

*

开本：787×1092 1/16 印张：14.25 字数：365 千字

2002 年 7 月第 1 版 2002 年 7 月第 1 次印刷 印数：5 000 册

ISBN 7-81077-201-5/TP·112 定价：21.00 元

前　言

随着科学技术的发展，电子产品的更新换代进一步加快，现代电子设计技术已进入一个全新的阶段。从中小规模的通用集成芯片构成电路系统，到应用微处理器、单片机构成数字系统，这一过程克服了中小规模集成电路在系统设计中的一些缺点，同时也为电子设计技术提供了一种软件设计的手段。然而，随着大规模和超大规模可编程逻辑器件在 EDA 技术支持下的广泛应用，使电子系统设计发生了质的变化。EDA 技术是从计算机辅助设计 CAD、计算机辅助制造 CAM、计算机辅助测试 CAT 和计算机辅助工程 CAE 等技术发展而来的。它以计算机为工具，设计者只需对系统功能进行描述，就可在 EDA 工具的帮助下完成系统设计。EDA 技术为电子产品的设计和开发缩短了时间，降低了成本，提高了系统的可靠性。

在 EDA 技术中，最为瞩目的是以现代电子技术为特征的逻辑设计仿真测试技术。该技术只需通过计算机就能对所设计的电子系统从不同层次的性能特点上，进行一系列准确测试和仿真；在完成实际系统的设计后，还能对系统上的目标器件进行边界扫描测试。高速发展的可编程逻辑器件又为 EDA 技术的不断进步奠定了坚实的物理基础。大规模可编程逻辑器件不但具有微处理器和单片机的特点，而且随着微电子技术和半导体制造工艺的进步，集成度不断提高，与微处理器、DSP、A/D、D/A、RAM 和 ROM 等独立器件之间的物理与功能界限正日趋模糊，嵌入式系统和片上系统(SOC)得以实现。以大规模可编程集成电路为物质基础的 EDA 技术打破了软硬件之间的设计界限，使硬件系统软件化。这已成为现代电子设计技术的发展趋势。

现代电子设计已经进入了数字化时代，电子设计的自动化程度将越来越高，传统的电子设计方法、工具和器件将在更大的程度上被 EDA 所取代。为了适应电子技术的发展和社会发展对人才的需求，本书介绍了最新的 EDA 技术和开发工具以及现代数字系统设计方法，目的是引导学生和设计人员从传统的通用集成电路的应用转向可编程逻辑器件的应用；从系统的硬件设计转向硬件、软件高度渗透的设计，以提高和拓宽数字系统的设计能力。

本书分为上、中、下三篇，即虚拟电子工作台、大规模可编程逻辑器件和现代数字系统设计。全书共十二章。

第一章介绍 EDA 技术的基本概念、研究范畴、基本特征和基本工具以及

EDA 技术的发展史。此外，还介绍了可编程 ASIC 的特点及发展趋势。

第二、三、四、五章为虚拟电子工作台（上篇），包括电子工作台（EWB）的特点、软件安装；EWB 基本界面及操作；仪器、仪表和元器件库的使用；EWB 的电路仿真及分析方法，如交直流分析、瞬态分析、傅里叶分析、零-极点分析、传递函数分析、温度扫描分析、灵敏度分析、蒙特卡罗分析等等。

第六、七、八、九、十章为大规模可编程逻辑器件（中篇），介绍可编程逻辑器件的分类、基本结构、编程元件和边界扫描测试技术。重点介绍 Altera 和 Lattice 公司的可编程逻辑器件的基本结构、基本资源和工作原理，以及 EDA 开发软件的使用方法和设计步骤，如 MAX+PLUS II 系统和 ISP Synari 系统。此外，又详述硬件描述语言 AHDL 和 ABEL - HDL 的设计方法。第十章给出了常用数字电路设计实例。

第十一、十二章为现代数字系统设计（下篇），包括数字系统的基本概念和基本结构；数字系统的设计特点和自顶向下的设计方法；描述数字系统的常用工具——算法流程图和 ASM 图。详细讨论了组成数字系统的两大部分，即数据处理单元和控制单元的设计以及实现方法。最后给出利用 EDA 开发工具进行数字系统的设计实例。

本书由包明、赵明富、陈渝光、雷建军、李太福、罗渝徽编写，由包明和赵明富任主编。按章节顺序，执笔人为：第一章：包明、罗渝徽；第二、三、四章：陈渝光、雷建军；第五、六章：赵明富、罗渝徽；第七、八章：赵明富、李太福；第九、十章：包明；第十一、十二章：包明、雷建军。全书由包明统稿。

本书承重庆大学博士生导师彭东林教授的审阅，对本书的编写提出了非常宝贵的意见，在此表示最衷心的感谢。

在本书的编写和出版过程中得到了北京航空航天大学出版社的大力支持，在此一并表示感谢。

EDA 技术正在不断发展，技术更新快，涉及面广，新的器件不断涌现。由于编者水平所限，书中的疏漏和错误恳请读者批评指正。

编　者

2001 年 9 月于重庆

目　录

第四章 EWB的元器件库

第五章 EWB的电路仿真及分析

中篇 大规模可编程逻辑器件

第六章 可编程逻辑器件概述

第九章 可编程逻辑器件的硬件描述语言

第十章 常用数字电路的设计实例

下篇 现代数字系统设计

第十一章 数字系统设计基础

第十二章 数字系统的实现

第一章 绪 论

1.1 EDA 技术

EDA 是 Electronics Design Automation(电子设计自动化)的缩写。它是随着集成电路和计算机技术的飞速发展应运而生的一种高级、快速、有效的电子设计自动化工具。EDA 工具是以计算机的硬件和软件为基本工作平台,集数据库、图形学、图论与拓扑逻辑、计算数学、优化理论等多学科最新成果研制而成的计算机辅助设计通用软件包。EDA 是电子设计技术的发展趋势,利用 EDA 工具可以代替设计者完成电子系统设计中的大部分工作。

数字系统的实现方法也经历了由分立元件、SSI、MSI 到 LSI、VLSI 以及 UVISI 的飞速发展过程。为了提高系统的可靠性与通用性,微处理器和专用集成电路(ASIC)逐渐取代了通用全硬件 LSI 电路。可编程逻辑器件(PLD)被大量地应用在 ASIC 的制作中,尤其是 FPLD/CPLD(现场可编程逻辑器件/复杂可编程逻辑器件)在 EDA 基础上的广泛应用,从某种意义上来说,是新的电子系统运转的物理机制又将回到原来的纯数字电路结构,是一种高层次的循环。它在更高层次上容纳了过去数字技术的优秀部分,是对 MUC(微控制器或单片机)系统的一种扬弃。特别是软/硬 IP 芯核产业的迅猛发展,嵌入式通用与标准 FPLD/CPLD 器件的出现,片上系统(System - On - Chip)已近在咫尺。以大规模可编程集成电路为物质基础的 EDA 技术将打破软硬件之间的设计界限,使硬件系统软件化,电子设计的技术操作和在系统构成的整体上将发生质的飞跃。EDA 技术带来了电子系统设计的革命性变化。

1.1.1 EDA 技术发展史

EDA 技术伴随着计算机、集成电路、电子系统设计的发展,经历了计算机辅助设计 CAD(Computer Assist Design)、计算机辅助工程设计 CAED(Computer Assist Engineering Design)和电子系统设计自动化 ESDA(Electronic System Design Automation)三个发展阶段,如图1.1.1 所示。

20 世纪 70 年代,随着中小规模集成电路的出现和应用,传统的手工制图设计印刷电路板和集成电路的方法已无法满足设计精度和效率的要求,人们开始将产品设计过程中高重复性的繁杂劳动,如布图、布线工作用二维平面图形编辑与分析的 CAD 工具代替。这就产生了第一代 EDA 工具。受当时计算机工作平台的制约,第一代 DEA 工具能支持的设计工作有限且性能比较差。

20 世纪 80 年代出现的第一个个人工作站(Apollo)计算机平台,推动了 EDA 工具的迅速发展。为了适应电子产品在规模和制作上的需要,出现了以计算机仿真和自动布线为核心技术的第二代 EDA 技术。具有自动综合能力的 CAE 工具代替了设计师的部分设计工作。其特点是以软件工具为核心,通过这些软件完成产品开发的设计、分析、生产、测试等各项工作。但是,大部分从原理图出发的 EDA 工具仍然不能适应复杂电子系统设计的要求,而且具体化

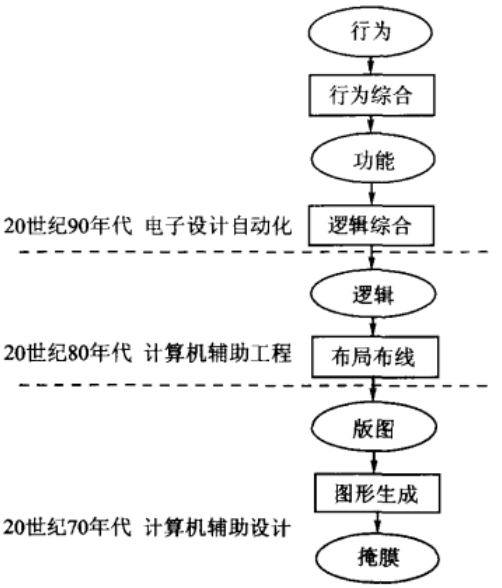


图 1.1.1 EDA 发展过程

的元件图形制约着优化设计。

20 世纪 90 年代，设计师逐步从使用硬件转向设计硬件，从电路级电子产品开发转向系统级电子产品开发。ESDA 工具是以系统级设计为核心，包括系统行为级描述与结构级综合、系统仿真与测试验证、系统划分与指标分配、系统决策与文件生成等一整套的电子系统设计自动化工具。第三代 EDA 技术的出现，极大地提高了系统设计的效率，使设计师开始实现“概念驱动工程”的梦想。设计师摆脱了大量的辅助设计工作，把精力集中于创造性的方案与概念构思上，从而极大地提高了设计效率，缩短了产品的研制周期。

1.1.2 EDA 与电子系统设计

传统的电子系统设计是采用搭积木式的方法进行设计，即由器件搭成电路板，由电路板搭成电子系统。数字系统最初的“积木块”是由固定功能的标准集成电路，如 74/54 系列(TTL)、4000/4500 系列(CMOS)芯片和一些固定功能的大规模集成电路构成。设计者只能根据需要选择合适的器件，并按照器件推荐的电路来组装系统。这种设计是一种“自底向上”的设计方法。这样设计出的电子系统所用元件的种类和数量均较多，体积、功耗大，可靠性差，不易修改。

随着半导体技术、集成技术和计算机技术的发展，电子系统的设计方法和设计手段发生了很大的变化。进入到 20 世纪 90 年代以后，电子设计自动化(EDA)技术的发展和普及给电子系统的设计带来了革命性的变化，特别是高速发展的 CPLD/FPGA 器件为 EDA 技术的不断进步奠定了坚实的物质基础，极大地改变了传统的数字系统设计方法、设计过程乃至设计观念。在传统的数字系统设计中，只有通过编程方式的两种途径，即微处理器的软件编程(如单片机)和特定器件的控制字配置(如 8255)来改变器件逻辑功能。这对器件引脚功能的硬件方式的任意确定是不可能的，而且对于系统设计只能通过设计电路板来实现系统功能。利用 EDA 工具通过对可编程器件芯片的设计来实现系统功能，这种方法称为基于芯片的设计方法。新的设计方法能够由设计者定义器件的内部逻辑和引脚，将原来由电路板设计完成的大

部分工作放在芯片的设计中进行。这样不仅可以通过芯片设计实现多种数字逻辑系统功能，而且由于引脚定义的灵活性，大大减轻电路图设计和电路板设计的工作量和难度，从而有效地增强了设计的灵活性，提高了工作效率；同时基于芯片的设计可以减少芯片的数量，缩小系统体积，降低能源消耗，提高系统的性能和可靠性。

可编程逻辑器件和 EDA 技术给今天的硬件系统设计者提供了强有力的工具，使得电子系统的设计方法发生了质的变化。传统的“固定功能集成块＋连线”的设计方法正逐步地退出历史舞台，而基于芯片的设计方法正在成为现代电子系统设计的主流。现在人们可以把数以亿计的晶体管，几万门、几十万门甚至几百万门的电路集成在一个芯片上。半导体集成电路也由早期的单元集成、部件电路集成发展到整机电路集成和系统电路集成。电子系统的设计方法也由过去的那种“Bottom-up”（自底向上）的设计方法改变为一种新的“Top-down”（自顶向下）设计方法。

现在，只要拥有一台计算机、一套相应的 EDA 软件和一片可编程逻辑器件芯片，在实验室里就可以完成数字系统的设计和生产。可以说，当今的数字系统设计已经离不开可编程逻辑器件和 EDA 设计工具。

1.1.3 EDA 软件平台

长期以来，大型的 EDA 系统都是运行在以 UNIX 为操作系统的工作站平台上。随着 PC 机性能的不断提高和 Windows 操作系统的逐步发展，世界上著名的 EDA 厂商如 Cadence Design Systems，Mentor Graphics，Synopsys，orCAD 和 Viewlogic Systems 等已先后推出了支持 PC－Windows 平台的 EDA 开发软件。在电路仿真和印刷电路板（PCB）设计时常用的 EDA 软件有：PSPICE，orCAD，FilterLab（模拟滤波器软件），CircuitMaker2000，PROTEL，Electronics workbench，PowerPCB EDA2000 等。

现代的 EDA 软件技术已突破了早期仅能进行 PCB 版图设计或者电路功能模拟的局限，以最终实现可靠的硬件系统为目标，配备了电子系统设计自动化的全部工具。如配置了多种能兼用和混合使用的逻辑描述输入方式：硬件描述语言文本输入法以及原理图输入法、波形输入法等；同时还配置了高性能的逻辑综合、优化和仿真模拟工具。目前可编程逻辑集成器件制造厂商推出了各种软件开发系统，为集成电路设计和制造提供了很好的 EDA 工具。表 1.1.1 给出了大规模可编程逻辑器件的常用 EDA 开发软件的特性。

表 1.1.1 可编程逻辑器件的 EDA 开发软件的特性

厂 商	EDA 软件名称	适用器件系列	输入方式
Vantis Lattice	Synario	MACH GAL，ispLSI，PLSI 等	原理图、ABEL、VHDL 文本等
Lattice	Expert	ispLSI，PLSI 等	原理图、VHDL 文本等
Altera	MAX＋PLUSⅡ	MAX，FLEX 等	原理图、波形图、VHDL、AHDL 文本等
Altera	Quartus	MAX，FLEX，APEX 等	原理图、波形图、VHDL、Verilog HDL 文本等
Actel	Actel Designer	SX 系列、MX 系列	原理图、VHDL 文本等
Vantis	Microsim	MACH	原理图
Xilinx	Alliance	Xilinx 各种系列	原理图、VHDL 文本等
Xilinx	Foundation	XC 系列	原理图、VHDL 文本等

1.2 EDA 技术的基本特征及工具

EDA 可以看作是电子 CAD 的高级阶段。在现代电子系统设计领域，EDA 技术已经成为电子系统设计的重要手段。无论是设计逻辑芯片还是数字系统，其设计作业的复杂程度都在不断增加，现今仅仅依靠手工进行数字系统设计已经不能满足要求，所有的设计工作都需要以计算机为工具，在 EDA 软件平台上进行。设计者只需完成对设计系统的功能描述，就可以由计算机自动地完成逻辑编译、逻辑简化、逻辑分割、逻辑综合及优化、逻辑布局布线、逻辑仿真，直至对于特定目标芯片的适配编译、逻辑映射和编程下载等工作。尽管目标系统是硬件，但整个系统设计和修改过程如同完成软件设计一样方便和高效。利用 EDA 的仿真测试技术，设计者可以预知设计结果，减少设计的盲目性，极大地提高设计的效率。

1.2.1 EDA 技术的研究范畴

EDA 技术研究的对象是电子设计的全过程，旨在帮助电子设计工程师在计算机上完成电路的功能设计、逻辑设计、性能分析、时序测试直至 PCB(印刷电路板)的自动设计。

与早期的电子 CAD 软件相比，EDA 软件的自动化程度更高，功能更完善，运行速度更快，而且操作界面友好，有良好的数据开放性和互换性，即不同厂商的 EDA 软件可相互兼容。因此，EDA 技术很快在世界各大公司、企业和科研单位得到广泛应用，并已成为衡量一个国家电子技术发展水平的重要标志。

EDA 技术的范畴贯穿于产品开发过程，以及电子产品生产的全过程中期望由计算机提供的各种辅助工作。从一个角度看，EDA 技术可粗略分为系统级、电路级和物理级 3 个层次的辅助设计过程；从另一个角度看，EDA 技术应包括电子线路设计的各个领域：即从低频电路到高频电路直至微波；从线性电路到非线性电路；从模拟电路到数字电路；从分立元件到集成电路的全部设计过程。EDA 技术的范畴和功能如图 1.2.1 所示。

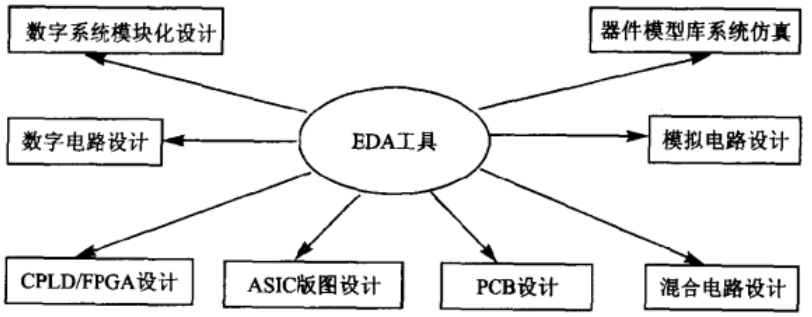


图 1.2.1 EDA 技术的范畴

1.2.2 EDA 技术的基本特征

现代 EDA 技术的基本特征是采用高级语言即硬件描述语言描述，具有系统级仿真和综合能力。下面介绍与这些基本特征有关的几个 EDA 技术的新概念。

1. 并行工程和"自顶向下"设计方法

根据美国防卫分析研究所 R－338 报告中的定义，所谓并行工程是指"一种系统化的、集成化的、并行的产品及相关过程的开发模式(相关过程主要指制造和维护)。这一模式使开发者从一开始就要考虑到产品生存周期的诸多方面，包括质量、成本、开发时间及用户的需求等等。"

"自顶向下"的设计方法是从系统级设计入手，在顶层进行功能方框图的划分和结构设计；在方框图一级进行仿真、纠错，并用硬件描述语言对高层次的系统行为进行描述；在系统一级进行功能验证，然后用逻辑综合优化工具生成具体的门级逻辑电路的网表，其对应的物理级可以是印刷电路板或专用集成电路。与"自底向上"的设计方法相比较，有利于在设计初期发现结构设计中的错误，提高设计的一次成功率，因而在现代 EDA 系统中被广泛采用。

2. 硬件描述语言

用硬件描述语言(HDL)进行电路与系统的设计是当前 EDA 技术的一个重要特征。与传统的原理图输入设计方法相比较，硬件描述语言更适合于规模日益增大的电子系统。它还是进行逻辑综合优化的重要工具。硬件描述语言使得设计者在比较抽象的层次上描述设计的结构和内部特征。它的突出优点是：语言的公开可利用性；设计与工艺的无关性；宽范围的描述能力；便于组织大规模系统的设计；便于设计的复用和继承等等。目前最常用的硬件描述语言有 VHDL 和 Verilog－HDL。它们都已经成为 IEEE 标准。另外还有一些 EDA 厂商自行开发的硬件描述语言，如 AHDL 和 ABEL。

3. 逻辑综合优化

逻辑综合功能将高层次的系统行为设计自动翻译成门级逻辑的电路描述，做到了设计与工艺的独立。优化则是对于上述综合生成的电路网表，根据布尔方程功能等效的原则，用更小更快的综合结果替代一些复杂的逻辑电路单元，根据指定的目标库映射成新的网表。

4. 开放性和标准化

框架是一种软件平台结构，为 EDA 工具提供了操作环境。框架的关键在于提供与硬件平台无关的图形用户界面以及工具之间的通信、设计数据和设计流程的管理等，此外还应包括各种与数据库相关的服务项目。任何一个 EDA 系统只要建立了一个符合标准的开放式框架结构，就可以接纳其他厂商的 EDA 工具一起进行设计工作。这样，框架作为一套使用和配置 EDA 软件包的规范，就可以实现各种 EDA 工具间的优化组合，并集成在一个易于管理的统一的环境之下，实现资源共享。

近年来，随着硬件描述语言等设计数据格式的逐步标准化，不同设计风格和应用的要求导致各具特色的 EDA 工具被集成在同一个工作站上，从而使 EDA 框架标准化。新的 EDA 系统不仅能够实现高层次的自动逻辑综合、版图综合和测试码生成，而且可以使各个仿真器对同一个设计进行协同仿真，进一步提高了 EDA 系统的工作效率和设计的正确性。

1.2.3 EDA 的基本工具

集成电路技术的进展不断对 EDA 技术提出新的要求，促进了 EDA 技术的发展。但是总的来说，EDA 系统的设计能力一直难以赶上集成电路技术的要求。EDA 工具的发展经历了两个大的阶段，即物理工具和逻辑工具阶段。现在 EDA 和系统设计工具正逐渐被理解成一个整体的概念——电子系统设计自动化。

物理工具用来完成设计中的实际物理问题，如芯片布局、印刷电路板布线等。另外它还能提供一些设计的电气性能分析，如设计规则检查。这些工作现在主要由集成电路厂家来完成。

逻辑工具是基于网表、布尔逻辑、传输时序等概念的。首先由原理图编辑器或硬件描述语言进行设计输入；然后利用 EDA 系统完成逻辑综合、仿真、优化等过程；最后生成物理工具可以接受的网表或 VHDL，VerilogHDL，AHDL，ABEL 的结构化描述。

现在人们已开发了大量的计算机辅助设计工具来帮助集成电路的设计，常见的 EDA 工具有编辑器、仿真器、检查/分析工具和优化/综合工具等，如图 1.2.2 所示。

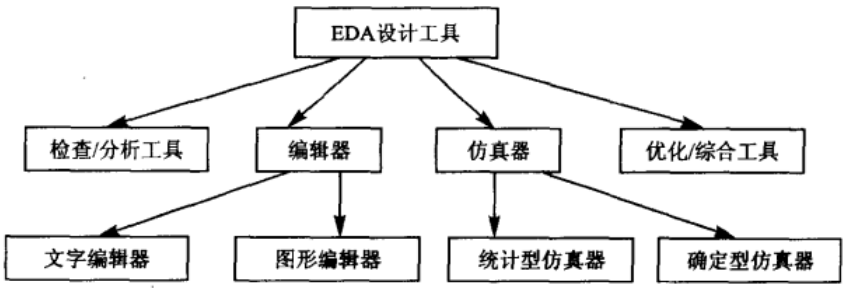


图 1.2.2 EDA 设计工具分类

1. 编辑器

编辑器包括文字编辑器和图形编辑器。在系统级设计中，文字编辑器用来编辑硬件系统的自然描述语言，在其他层次用来编辑电路的硬件描述语言文本。在数字系统中的门级、寄存器级以及芯片级，所用的描述语言通常为 VHDL，Verilog - HDL，AHDL 和 ABEL。在模拟电路级，硬件描述语言通常为 SPICE 的文本输入。

图形编辑器可用于硬件设计的各个层次。在版图级，图形编辑器用来编辑表示硅工艺加工过程的几何图形。在高于版图层次的其他级，图形编辑器用来编辑硬件系统的方框图、原理图等。原理图输入工具至少应包括以下 3 个组成部分：

① 基本单元符号库，主要包括基本单元的图形符号和仿真模型。硬件设计者除了采用基本单元和标准单元之外，还应该能够使用原理图编辑器建立自己专用的图形符号以及相应的仿真模型，加到基本单元符号库中，供设计时使用和调用。

② 原理图编辑器的编辑功能。

③ 产生网表的功能。

2. 仿真器

仿真器又称模拟器，主要用来帮助设计者验证设计的正确性。在硬件系统设计的各个层次都要用到仿真器。在数字系统设计时，硬件系统用数字逻辑器件以及它们之间的互联来表示。仿真器的用途是确定系统的输入/输出关系，所采用的方法是把每一个数字逻辑器件映射为一个或几个进程，把整个系统映射为由进程互联构成的进程网络。这种由进程互联组成的网络就是设计的仿真模型。

3. 检查/分析工具

在集成电路设计的各个层次都会用到检查/分析工具。在版图级，必须用设计规则检查/分析工具来保证版图所表示的电路可以被可靠地制造出；在逻辑门级，检查/分析工具可以用来检查是否有违反扇出规则的连接关系；时序分析器可以用来检查最坏情形时电路中的最大

和最小延时。

4. 优化/综合工具

优化/综合工具用来把一种硬件描述转换为另一种描述，这里的转换过程通常伴随着设计的某种改进。在逻辑门级，可以用逻辑最小化来对布尔表达式进行简化；在寄存器级，优化工具可以用来确定控制序列和数据路径的最优组合。各个层次的综合工具可以将硬件的高层次描述转换为低层次描述，也可以将硬件的行为描述转换为结构描述。

1.3 可编程 ASIC 特点及发展趋势

可编程 ASIC 特别是现代可编程 ASIC(CPLD,FPGA)的出现，使得电子设计工程师或科研人员有条件在实验室内快速、方便地开发专用集成电路，这些专用集成电路往往就是一个复杂的数字系统。因此，可以说可编程 ASIC 给现代电子系统的设计带来了极大的变革。

1.3.1 专用集成电路 ASIC 简介

ASIC 是 Application Specific Integrated Circuits(专用集成电路)的缩写。它是面向专门用途的电路，以此区别于标准逻辑(Standard Logic)、通用存储器、通用微处理器等电路，是专门为一个用户设计和制造的。换言之，它是根据某一用户的特定要求，能以低研制成本、短交货周期供货的全定制、半定制集成电路。ASIC 的概念早在 20 世纪 60 年代就有人提出，但由于当时设计自动化程度低，加上工艺技术、市场和应用条件均不具备，因而没有得到适时发展。进入 20 世纪 80 年代后，随着半导体集成电路的工艺技术、制造技术、设计技术、测试评价技术的发展，集成度的不断提高，为开发周期短、成本低、功能强、可靠性高以及专利性与保密性好的专用集成电路创造了必要而充分的发展条件，并很快形成了用 ASIC 取代中、小规模集成电路来设计电子系统或整机的技术热潮。

目前 ASIC 在总的 IC 市场中的占有率已发展到近三分之一，在整个逻辑电路市场中的占有率已超过一半。与通用集成电路相比，ASIC 在构成电子系统时具有以下几个方面的优点：

① 缩小体积，减轻重量，降低功耗。

② 提高可靠性。用 ASIC 芯片进行系统集成后，外部连线减少，可靠性明显提高。

③ 易于获得高性能。ASIC 针对专门的用途而特别设计，是系统设计、电路设计和工艺设计的紧密结合。这种一体化的设计能得到前所未有的高性能系统。

④ 可增强保密性。电子产品中的 ASIC 芯片对用户来说相当于一个“黑盒子”。

⑤ 在大批量应用时，可显著降低系统成本。

ASIC 按功能的不同可分为数字 ASIC、模拟 ASIC 和微波 ASIC；按使用材料的不同可分为硅 ASIC 和砷化镓 ASIC。一般地说，数字、模拟 ASIC 主要采用硅材料；微波 ASIC 主要采用砷化镓材料。砷化镓具有高速、抗辐射能力强、寄生电容小和工作温度范围宽等优点，目前已在移动通信、卫星通信等方面得到广泛应用。

按照设计方法的不同，ASIC 可分为全定制和半定制两类。全定制法是一种基于晶体管级的设计方法。对于某些性能要求很高、批量较大的芯片，一般采用全定制法设计。半定制法是一种约束性设计方法，约束的主要目的是简化设计，缩短设计周期，提高芯片的成品率。先以最短的时间设计出芯片，在占领市场的过程中再予以改进，进行二次开发。目前广泛采用的

半定制设计方式有：门阵列法、标准单元法和可编程逻辑器件法。

1. 全定制法

全定制法是一种基于晶体管级的设计方法。设计者必须使用版图编辑工具从晶体管的版图尺寸、位置及互联线开始亲自设计，以期得到ASIC芯片的最优性能。

运用全定制法设计芯片，对芯片的功能、性能、面积和成本确定后，设计人员要对芯片结构、逻辑、电路等进行精心的设计，对不同的方案进行反复比较，对单元电路的结构、晶体管的参数要反复地模拟优化。在版图设计时，设计人员要手工设计版图并精心地布局布线，以获得最佳的性能和最小的面积。版图设计完成后，要进行完整的检查、验证，包括设计规则检查、电学规则检查、连接性检查、版图参数提取、电路图提取、版图与电路图一致性检查等。最后，通过模拟，才能将版图转换成标准格式的版图文件交与厂家制造芯片。例如半导体厂家推出的新的微处理器芯片，为了提高芯片的速度，设计时需采用最佳的随机逻辑网络，且每个单元都必须精心设计；另外还要精心地布局布线，将芯片设计得最紧凑，以节省每一小块面积，降低成本。

由此可见，采用全定制法可以设计出最高速度、最低功耗和最省面积的芯片，但设计的周期很长(一般1年)，设计成本较高，只适用于对性能要求很高(如高速芯片)或批量很大的芯片(如存储器、通用芯片)，由IC厂家一次性制造出来。

2. 门阵列法

门阵列设计法又称“母片”法，是最早开发并得到广泛应用的ASIC设计技术。母片是IC工厂按照一定规格事先生产的半成品芯片。在这个芯片上制作了大量按一定规则排列的门单元，并排列成阵列形式。这些单元依照要求相互连接在一起即可实现不同的电路要求。母片完成了绝大部分芯片工艺，只留下一层或两层金属铝连线的掩膜，需要根据用户电路的不同而定制。

门阵列设计方法涉及的工艺少，设计软件一般都具有较高的自动化水平，设计制造周期短，设计成本低。但门的利用率不高，芯片面积较大，母片上制造好的晶体管都是固定尺寸的，不利于设计高性能的芯片。

3. 标准单元法

标准单元设计法又称库单元法。它是由IC厂家在芯片版图一级预先设计好一批具有一定逻辑功能的单元，这些单元在功能上覆盖了中小规模标准IC的功能，并以库的形式放在EDA工具中。设计时可根据需要选择库中的标准单元构成电路，然后调用这些标准单元的版图，并利用自动布局布线软件完成电路到版图一一对应的最终设计。

相对于全定制设计法，标准单元法设计的难度和设计周期都小得多，而且也能设计出性能较高、面积较小的芯片。与门阵列法相比，标准单元法设计的电路性能、芯片利用率以及设计的灵活性均比门阵列好，既可用于设计数字ASIC，又可用于设计模拟ASIC。但标准单元库的投资较大，而且芯片的制作需要全套的掩膜版和全部工艺过程，因此生产周期及成本均比门阵列高。它适用于性能指标较高而生产批量又较大的芯片设计。

4. 可编程逻辑器件法

可编程逻辑器件是ASIC的一个重要分支，与前面介绍的几类ASIC不同，它是一种已完成了全部工艺制造、可直接从市场上购得的产品。用户只要购得通用的可编程器件，由自己通过EDA工具软件对器件进行功能配置，实现用户的专用要求即可。为了与由IC工厂专门掩

膜制造的 ASIC 相区别，又称它为可编程 ASIC。前面 3 种方法设计的 ASIC 芯片都必须到 IC 厂家去加工制造才能完成，设计制造周期长，而且一旦有了错误，需重新修改设计和制造，成本和时间要大大增加；而采用可编程逻辑器件，设计者在实验室即可设计和制造出芯片，而且可反复编程，修改错误，这样方便了设计者。

可编程逻辑器件发展到现在，规模越来越大，功能越来越强，价格越来越便宜，相配套的 EDA 软件越来越完善，因而深受设计者的喜爱。

1.3.2 可编程 ASIC 的主要特点

可编程 ASIC 是指由用户编程来实现所需功能的专用集成电路，按照结构的复杂程度不同，大致可分为以下几类：

① 简单可编程 ASIC，如 PAL 和 GAL；

② 复杂可编程 ASIC，如 CPLD；

③ 现场可编程 ASIC，如 FPGA。

尽管这三种可编程器件其结构和性能不尽相同，但有一个共同点就是都由用户通过编程来决定芯片的最终功能，因此被统称为可编程 ASIC。可编程 ASIC 与掩膜 ASIC 相比具有以下特点。

1. 缩短研制周期

可编程 ASIC 相对于用户而言，可以按一定的规格型号像通用器件一样在市场上买到，其 ASIC 功能的实现是完全独立于 IC 工厂的，由用户在实验室或办公室就可完成，因此不必像掩膜 ASIC 那样花费样片制作等待时间。由于采用先进的 EDA 工具，可编程 ASIC 的设计与编程均十分方便和有效，整个设计通常只需几天便完成，缩短了产品研制周期，有利于产品的快速上市。

2. 降低设计成本

制作掩膜 ASIC 的前期投资费用较高，动辄数万元，只有生产批量很大的情况下才有价值。这种设计方法还需承担很大的风险，若不能一次成功，需要修改，则全套掩膜便不能再用，巨额的设计费用将付之东流。采用可编程 ASIC 为降低投资风险提供了合理的选择途径。它不需掩膜制作费用，在设计的初期或在小批量的试制阶段，其平均单片成本远低于门阵列。如果要转入大批量生产，由于已用可编程 ASIC 进行了原型验证，也比直接设计掩膜 ASIC 费用小、成功率高。

3. 提高设计灵活性

可编程 ASIC 是一种由用户编程实现芯片功能的器件，与由工厂编程的掩膜 ASIC 相比，具有更好的设计灵活性。第一，可编程 ASIC 在设计完成后可立即编程进行验证，有利于及早发现设计中的问题，完善设计；第二，可编程 ASIC 中大多数器件均可反复多次编程，为设计修改和产品升级带来了方便；第三，基于 SRAM 开关的现场可编程门阵列 FPGA 和基于 E^2CMOS 工艺的在系统可编程逻辑器件 isPLD 具有动态重构特性，使系统设计中引入了“软硬件”的全新概念，使得电子系统具有更好的灵活性和自适应性。

1.3.3 可编程 ASIC 发展趋势

历史上，以可编程逻辑器件为代表的可编程 ASIC 经历了从简单到复杂，从小规模到大规

模，从低速到高速器件的发展演变过程。

① 20世纪70年代初，熔丝编程的PROM和PLA器件是最早的可编程逻辑器件。

② 20世纪70年代末，AMD公司开始推出PAL器件。

③ 20世纪80年代初，Lattice公司发明了电可擦写的、比PAL器件使用更灵活的GAL器件。

④ 20世纪80年代中期，Xilinx公司提出现场可编程的概念，同时生产了世界上第一片FPGA器件。同一时期，Altem公司推出了EPLD器件，较GAL器件有更高的集成度，可以用紫外线或电擦除。

⑤ 20世纪80年代末，Lattice公司又提出了在系统可编程的概念，并且推出了一系列具备在系统可编程能力的CPLD器件。

到了20世纪90年代，可编程ASIC技术进入了飞速发展的时期，现代电子系统的设计为可编程ASIC器件提供了一个广阔的应用领域。可编程ASIC市场的增长主要来自大容量的可编程逻辑器件CPLD和FPGA，其未来的发展趋势呈现为以下几个方面。

1. 向高密度、大规模的方向发展

电子系统的发展必须以电子器件为基础，随着集成电路制造技术的发展，可编程ASIC器件的规模不断地扩大，从最初的几百门到现在的上百万门。目前，高密度的可编程ASIC产品已经成为主流器件，可编程ASIC已具备了片上系统(SOC)集成的能力，产生了巨大的飞跃。这也促使了工艺的不断进步，而每次工艺的改进，可编程ASIC器件的规模都将有很大的扩展。由于高密度、大容量的可编程ASIC的出现，给现代电子系统(复杂系统)的设计与实现带来了巨大的帮助。

2. 向系统内可重构的方向发展

系统内可重构是指可编程ASIC在置入用户系统后仍具有改变其内部功能的能力。采用系统内可重构技术，使得系统内硬件的功能可以像软件那样通过编程来配置，从而在电子系统中引入“软硬件”的全新概念。它不仅使电子系统的设计、产品性能的改进和扩充变得十分简便，还使新一代电子系统具有极强的灵活性和适应性，为许多复杂信号的处理和信息加工的实现提供了新的思路和方法。

按照实现的途径不同，系统内重构可分为静态重构和动态重构两类。对基于E^2ROM或快速擦写技术的可编程器件，系统内重构是通过在系统编程ISP(In System Programmability)技术实现的，是一静态逻辑重构。另一类系统重构即动态重构，是指在系统运行期间，根据需要适时地对芯片重新配置以改变系统的功能，可由基于SRAM技术的FPGA实现。可编程ASIC的系统内可重构特性有着极其广泛的应用前景，近年来在通信、航天、计算机硬件系统、程序控制、数字系统的测试诊断等多方面获得了较好的应用。

3. 向低电压、低功耗的方向发展

集成技术的飞速发展，工艺水平的不断提高，节能潮流在全世界的兴起，也为半导体工业提出了降低工作电压的发展方向。可编程ASIC产品作为电子系统的重要组成部分，也不可避免地向3.3 V—2.5 V—1.8 V的标准靠拢，以便适应其他数字器件，扩大应用范围，满足节能的要求。

4. 向高速可预测延时器件的方向发展

可编程ASIC产品能得以广泛的应用，与之灵活的可编程性分不开；另一方面时间特性也

是一个重要的原因。作为延时可预测的器件，可编程 ASIC 的速度在系统中的作用巨大。在当前的系统中，由于数据处理量的激增，要求数字系统有大的数据吞吐量，加之多媒体技术的迅速发展，更多的是图像的处理，相应的要有高速的硬件系统，而高速的系统时钟是必不可少的条件。可编程 ASIC 产品也必然向高速发展。另外，为了保证高速系统的稳定性，可编程 ASIC 器件的延时可预测性也是十分重要的。用户在进行系统重构的同时，担心的是延时特性会不会因重新布线的改变而改变，否则将导致系统重构的不稳定性，这对于庞大而高速的系统而言将是不可想象的，其带来的损失将是巨大的。因此，为了适应未来高速电子系统的要求，可编程 ASIC 的高速可预测延时也是一个发展趋势。

5. 向混合可编程技术方向发展

可编程 ASIC 的广泛应用使得电子系统的构成和设计方法均发生了很大的变化。但是迄今为止，有关可编程 ASIC 的研究和开发的大部分工作基本上都集中在数字逻辑电路上，在未来几年里，这一局面将会有所改变，模拟电路及数模混合电路的可编程技术将得到发展。

国外已有几家公司开展了这方面的研究，并且推出了各自的模拟与数模混合型的可编程器件。例如 Lattics 公司开发的 EPAC(可编程模拟电路)和 International Microelectronic Products 公司开发的 EPAC。这种芯片上的各种模拟电路的功能也是由用户编程来决定的，如可编程增益放大器、可编程比较器、可编程多路复用器、可编程数模转换器、可编程滤波器和跟踪保持放大器等。用户可利用公司自己专门提供的开发工具来完成原型设计，确定器件配置，再把设计好的配置数据下载到芯片上，就可以通过它们去控制优化的模拟开关，进而把芯片上的各种模拟电路互联起来。此外，美国 Motorola 公司也于近期推出了一种基于开关电容技术的现场可编程模拟阵列 MPAA020 及相应的开发软件 EasyAnalog。这种器件也和 EPAC 一样，能够通过编程来实现一些常用的模拟电路的功能。

可编程模拟 ASIC 是今后模拟电子线路设计的一个发展方向。它们的出现使得模拟电子系统的设计也和数字系统设计一样变得简单易行，为模拟电路的设计提供了一个崭新的途径。

可编程 ASIC 是一门正在发展的技术，随着工艺和结构的改进，可编程 ASIC 的集成度将进一步提高，性能将进一步完善，成本将逐渐下降，在现代电子系统设计中将起到越来越大的作用，并且得到更加广泛的应用。

上篇　虚拟电子工作台

第二章　电子工作台(EWB)概述

2.1　电子工作台(EWB)简述

从事电子产品设计、研制的人员，经常需要对所设计的电路进行实物模拟和调试。一方面是为了验证设计的电路是否能达到设计要求的技术指标，另一方面通过改变电路中元器件的参数，使整个电路达到最佳性能。过去的电路设计过程，常常需要制作模拟实验板，在实验板上用实际元器件安装和调试。取得数据后，再来修改原设计的电路参数，反复多次直至达到设计要求。由于受工作场地、仪器仪表和元器件品种、数量的限制，使产品的研制开发无法及时完成，增加了研发周期和产品成本。随着计算机技术的发展，利用计算机辅助设计和仿真，对电路及系统设计不需要昂贵的实验设备，通过计算机提供的安全有效的设计环境，很方便地使得电路结构及设计观念得到修正。这样设计者直接用计算机模拟、分析、验证和调试，可以快速地反映出设计的电路性能指标，无论是教学还是科研都可以提高工作效率以及节省产品的开发成本和时间。

加拿大 Interactive Image Technologies 公司于 20 世纪 90 年代初推出了专门用于电子线路仿真的“虚拟电子工作台”(Electronics Workbench)软件，可以将不同类型的电路组合成混合电路进行仿真。与其他的电路仿真软件相比较，Electronics Workbench 软件具有界面直观、操作方便等优点。它改变了有些电路仿真软件输入电路采用文本方式的不便之处，创建电路、选用元器件和测试仪器等均可以直接从屏幕图形中选取，并且用画电路图的方式建立电路，而且测试仪器图形与实物外形基本相似。目前已在电子工程设计、电子类课程教学和实验等领域得到了广泛的应用。

EWB 软件还是一种非常优秀的电子技术实践训练工具。要掌握电子技术，不仅要学好理论知识，而且还要通过实际操作来加深对内容的理解。EWB 软件作为电子类相关课程的辅助教学和实验手段，不仅可以弥补实验仪器、元器件缺乏带来的不足，而且避免了原材料消耗和仪器损坏等因素，可以帮助学生更快、更好地掌握课堂讲述的内容，加深对基本概念、原理的理解，弥补课堂理论教学的不足，而且通过电路仿真，可以熟悉常用电子仪器的测量方法，进一步培养学生的综合分析能力、排除故障的能力和开发、创新能力。

2.2 电子工作台(EWB)的特点

EWB 软件是以 SPICE3F5 为模拟软件的核心，并增强了其在数字及混合信号仿真方面的功能。它可为设计者提供所需的各种元器件和仪器仪表，进行计算机辅助设计、模拟及布局，以产生印刷板层次的电路。EWB 软件具有如下主要特点：

① EWB 提供交互式的人机图形界面。绘制电路图需要的元器件、电仿真需要的测试仪器均可直接从屏幕上选取，并且选用的元器件和仪器与实际情况非常相近。

② EWB 具有完整的模拟和数字混合仿真的功能，其元器件库不仅提供了数千种电路元器件供选用，而且还提供了各种元器件的理想值，同时也可以新建或扩充已有的元器件库，大大方便了使用者。

③ EWB 具有下拉式的电路编辑功能表，绘制电路及元件输入变得简易快速。在输出信号的观察上，具备即时波形显示的功能。

④ EWB 具有虚拟的仪器仪表设备，包含波形函数产生器、万用表、示波器及逻辑分析等。

⑤ EWB 提供了较为详细的电路分析手段，不仅可以完成电路的瞬态分析和稳态分析、时域和频域分析、器件的线性和非线性分析、电路的噪声和失真分析等常规电路分析方法，而且还提供了离散付里叶分析、电路零-极点分析、交直流灵敏度分析和电路容差分析等共计 14 种电路分析方法。

2.3 EWB 软件设计过程及软件安装

1. EWB 软件设计过程

EWB 设计流程如图 2.3.1 所示。在概念的形成阶段，利用 EWB 可启发设计想法及先期构思。然后进一步将设计构思不断修正完善，通过 EWB 仿真分析，确定完整的电路结构及参数。当电路设计完成并准备实做时，可利用 EWB 输出文件即网表文件送至 PCB 排版布局软件，产生印制线路图，如 EWB 的 Layout 软件、OrCAD、Protel 及 Tango 等商用布线软件。

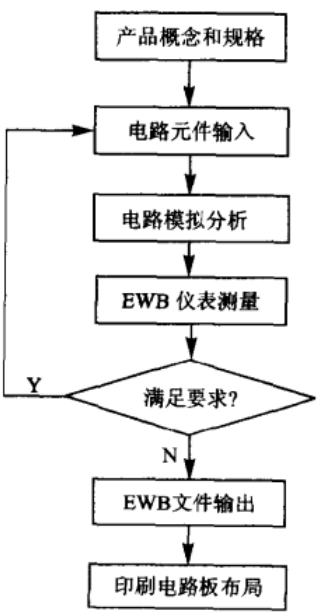


图 2.3.1 EWB 的设计流程

EWB 软件在电路设计中主要经历四个阶段：

① 输入阶段：输入电路元件、绘制电路图、设定元件参数。EWB 将会读入所设计电路的相关信息。

② 设定阶段：EWB 将会建立及检查一组包含电路描述的信息结构。

③ 分析阶段：选定适当的分析并进行运行，此阶段几乎占去 CPU 的所有时间，根据选定的分析和仿真算法，产生电路方程式并求解，然后提供所有资料直接输出或作图形处理。

④ 输出阶段：借助仪器仪表来观察输出波形和测量具体数据，或者由 Analysis 功能分析仿真结果；并且可产生

各种印制线路板排版软件能接收的网表文件。

2. 软件安装

随着计算机软件的发展，特别是 Windows 操作系统软件的广泛应用，EWB 软件也是从 DOS 版发展成可在 Windows 下运行的版本，最新 EWB 5.0 版本适用于 Windows 3.1 以上、486 以上 PC、至少 16 M RAM 及 20 M 的硬盘等工作环境。其软件安装过程根据屏幕提示信息进行，即确定程序安装位置、工作目录、输入用户信息和序列号。由于 EWB 5.0 版本的软件带有“硬件狗”，所以在进行软件安装和运行时，必须把它安装在计算机的并行输出口上。在安装完毕，启动 EWB 图标后，其工作界面如图 2.3.2 所示。

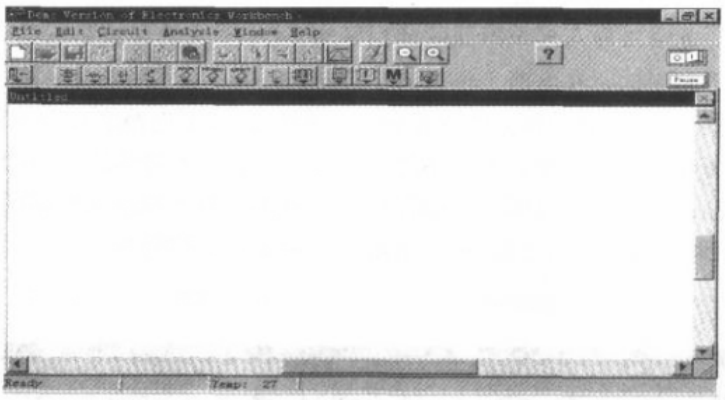


图 2.3.2　Workbench 工作界面

第三章　EWB基本界面及操作

3.1　EWB窗口界面

启动EWB 5.0后，将出现如图3.1.1所示的主窗口。从图中可看出，EWB模仿了一个实际的电子实验台。此窗口显示多个不同的使用区域，其中最大的区域是电路工作区，在这里可以进行电路的编辑和测试。在电路工作区的下方是描述窗口，可用来对电路进行注释和说明。工作区的上面是菜单栏、工具栏和元器件库栏。菜单栏提供了电路文件的存取、编辑、分析及实验所需的各种命令。工具栏包含了常用的操作命令按钮。元器件库栏包含了电路实验所需的各种元器件与测试仪器。通过鼠标器操作即可实现各种命令和绘制电路图。单击“启动/停止”按钮可以方便地控制实验的进程，进行电路仿真模拟和各种分析。

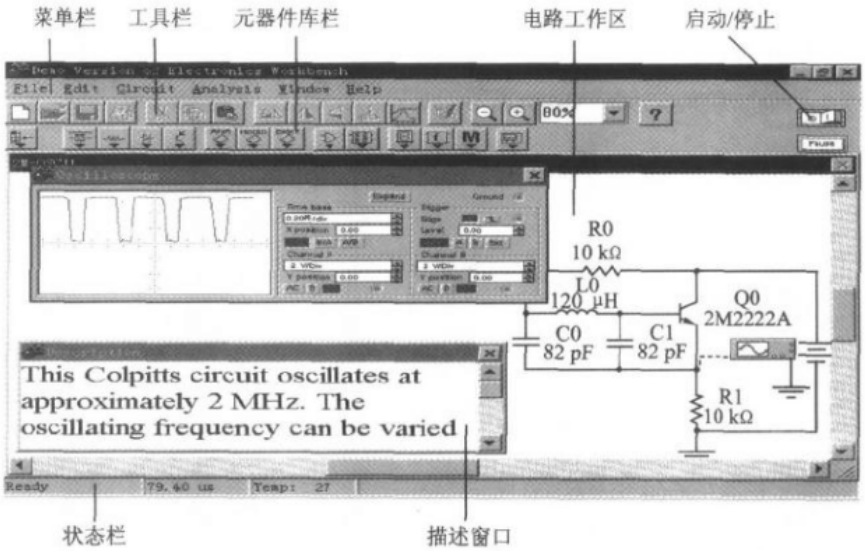


图3.1.1　主窗口界面

① 菜单栏：如图3.1.2所示。主要操作可归类为文件(File)、编辑(Edit)、电路(Circuit)及分析(Analysis)。其中文件和编辑与一般Windows软件的用法类似，电路和分析将在后面章节中叙述。

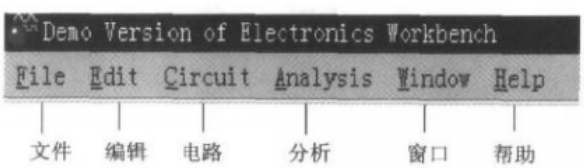


图3.1.2　菜单栏

② 工具栏：如图 3.1.3 所示。其中，一类为电路工具栏，包含用于编辑电路设计所需的常用操作命令；另一类为元器件库栏，将在后面叙述。

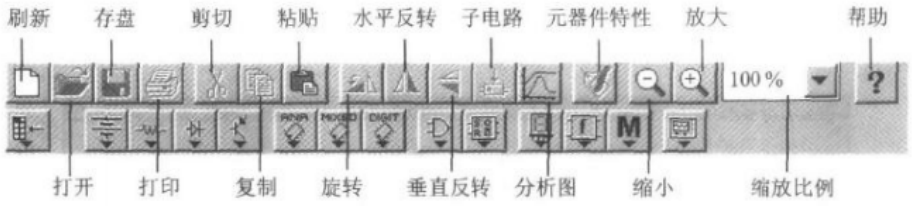


图 3.1.3　工具栏

③ 应用窗口：包含电路工作区、描述窗口、状态栏及子电路窗口。电路工作区是使用者进行电路设计的窗口；描述窗口是对电路进行注释和描述；状态栏是显示鼠标所指处元件或仪表的名称，还可显示仿真中的现状及运行时间。

3.2　电路的创建与运行

了解了 EWB 窗口界面的内容后，本节将讨论如何利用 EWB 进行电路的建立及测试。

3.2.1　元器件的操作

1. 元器件的选用

选用元器件时，首先在元器件库栏中单击包含该元器件的图标，打开该元器件库，然后从元器件库中将该元器件拖拽至电路工作区。在电路中需要选中某个元器件时，只要使用鼠标的左键单击该元器件，该元器件以红色显示来表示被选中。

2. 元器件的移动、旋转

要移动一个元器件，先选中该元器件，然后拖拽该元器件即可。要移动一组元器件，先画出一个矩形区域，选中了该矩形区域内的元器件，然后拖拽被选中的元器件，即可实现元器件移动、编辑。

要对元器件进行旋转或反转操作，同样应该先选中该元器件，然后使用工具栏中的“旋转、垂直反转、水平反转”等按钮，或者选择 Circuit 菜单中的旋转命令。

3. 元器件复制、删除

对选中的元器件，使用 Edit 菜单中的 Cut，Copy 和 Paste，Delete 等命令可实现元器件的复制、移动、删除等操作。此外，直接将元器件（已打开状态）拖拽回元器件库也可以实现删除操作。

4. 元器件标签、编号、数值、模型参数设置

在选中元器件后，再按下工具栏中的器件特性按钮，或者选择菜单命令会弹出器件特性对话框。它具有多种选项可供设置，包括 Label（标识）、Models（模型）、Value（数值）、Fault（故障设置）、Display（显示）、Analysis Setup（分析设置）等内容。

Label 选项用于设置元器件的 Label（标识）和 Reference ID（编号）。其编号通常由系统自动分配，必要时可以修改，但必须保证编号的惟一性。有些元器件没有编号，如连接点、接地、电压表、电流表等。

Value(数值)选项用于设置元器件的数值大小。对于比较复杂的元器件,会出现 Models(模型)选项。图 3.2.1 所示为晶体管的特性。一般是缺省设置(default),通常为 ideal(理想),能够满足多数情况下的分析要求。如果对分析精度有特殊要求,则可以选择具有具体型号的器件模型。

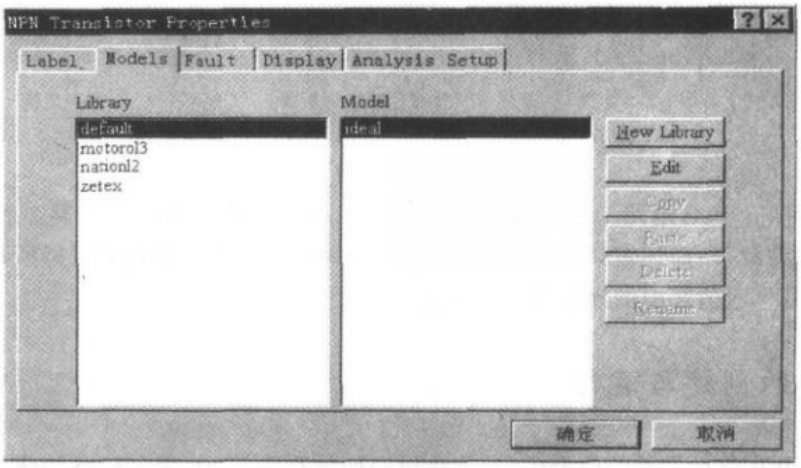


图 3.2.1 晶体管的特性

Fault(故障)选项可供人为设置元器件的隐含故障。图 3.2.2 为某个电容的故障设置情况。Display(显示)选项用于设置 Label,Models,Reference ID 的显示方式;Analysis Setup(分析设置)选项用于设置电路的工作温度等有关参数;Node(节点)选项用于设置与节点有关的参数。

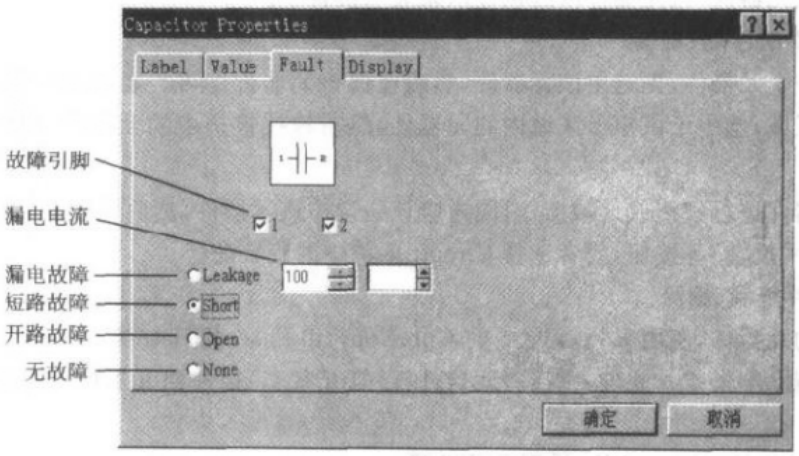


图 3.2.2 故障设置选项对话框

5. 电路图选项的设置

选择 Circuit/Schematic Options(电路/电路图选项)菜单命令,可弹出如图 3.2.3 所示的对话框。该选项用于设置与电路图显示方式有关的一些选项,包括栅格设置、布线设置,设置标号、数值、元器件字体、字号等的显示方式。

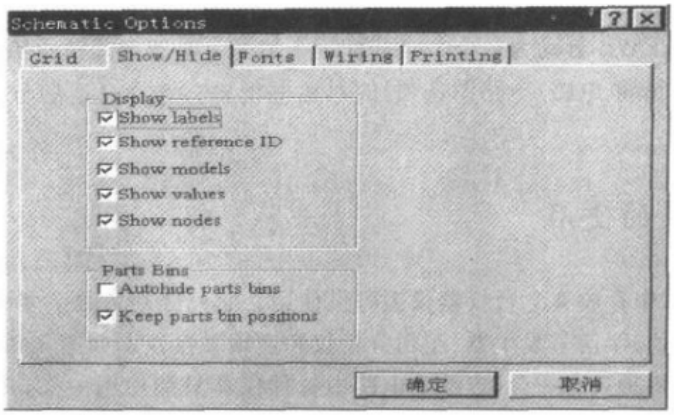


图 3.2.3　电路图选项设置对话框

3.2.2　导线的操作

1. 导线的连接

首先将鼠标器指向元器件的端点使其出现一个小黑圆点；按下鼠标左键并拖拽出一根导线，拖向另一个元器件的端点使其出现一个小黑圆点处；释放鼠标左键，则导线连接自动完成。

2. 连线的删除与改动

将鼠标器指向元器件与导线的连接点，出现一个小黑圆点；按下鼠标左键拖拽该圆点使其导线离开元器件的端点；释放左键，自动完成导线连线的删除。也可先选中导线连线，按下“Delete”键即可。

3. 在线路上插入元器件

可以直接拖拽元器件放置在导线上，然后释放，即可插入电路连线中。

4. 改变导线的颜色

可以将导线设置为不同的颜色，有助于对电路图的识别。首先双击该导线，立即弹出 Wire Properties 对话框，选择 Schematic Options 选项，然后选择合适的颜色，如图 3.2.4 所示。

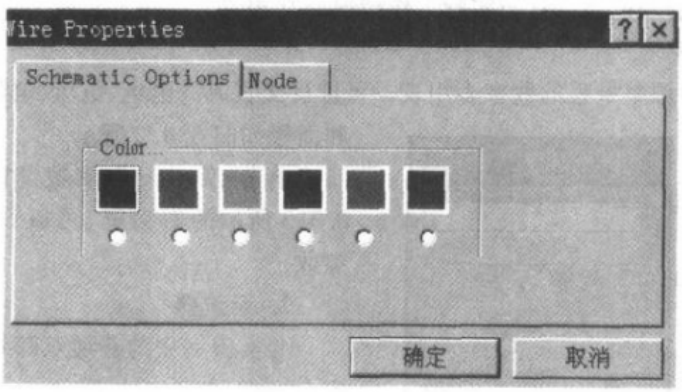


图 3.2.4　导线颜色设置对话框

5. 节点及其标识、编号和颜色

在连接电路时，EWB自动为每个节点分配一个编号。是否显示节点编号由菜单命令Circuit/Schematic Options(电路/电路图选项)的对话框设置。选择合适的节点颜色同选择导线颜色的方法相似。

3.3 仪器仪表的使用

EWB的仪器库中存放有七台仪器仪表可供使用。它们分别是数字多用表、函数信号发生器、示波器、波特图仪、字信号发生器、逻辑分析仪和逻辑转换仪。如图3.3.1所示为这些仪器的图标。这些仪器每种只有一台，连接到电路中每种仪器只能使用一次，并以图标方式显示。当需要观察测试数据和波形以及设置仪器参数时，可以双击仪器图标打开仪器面板。此外EWB还提供电压表和电流表。这两种电表的数量是没有限制的，存放在指示元件库中可供多次选用。在电路中对仪器的选用、删除和连接与使用元器件的方法相似。

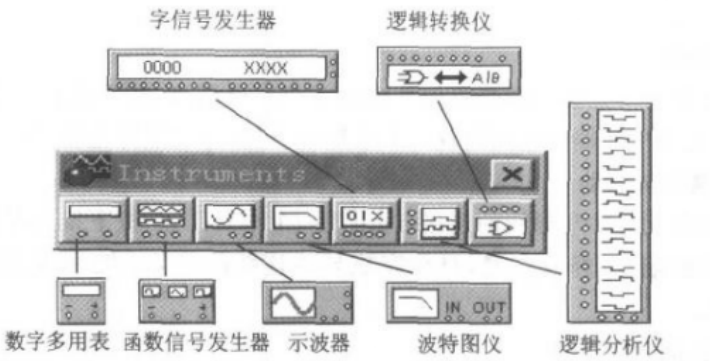


图3.3.1 仪器图标

3.3.1 模拟仪表的使用

模拟仪表主要包括数字多用表、函数信号发生器、示波器、波特图仪以及电压表和电流表。仪器仪表在接入电路并运行启动后，若改变其在电路中的连接处，则显示的数据和波形也相应改变，而不必重新启动电路。这与实际工作情况非常相似。

1. 数字多用表

这是一种自动调整量程的数字多用表。可进行交直流的电压、电流、电阻的测量。其图标和面板如图3.3.2所示。

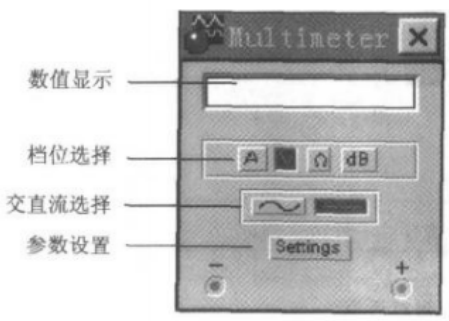


图3.3.2 数字多用表

单击“Settings”(参数设置)按钮，弹出如图3.3.3所示的对话框，可以设置多用表内部的参数。

2. 示波器

与实验室中的普通双踪示波器的使用方法完全一样，其面板如图3.3.4所示。为了能够更细致地观察波形，单击示波器面板“Expend”(扩展)按钮，可以将示波器面板进一步展开，如

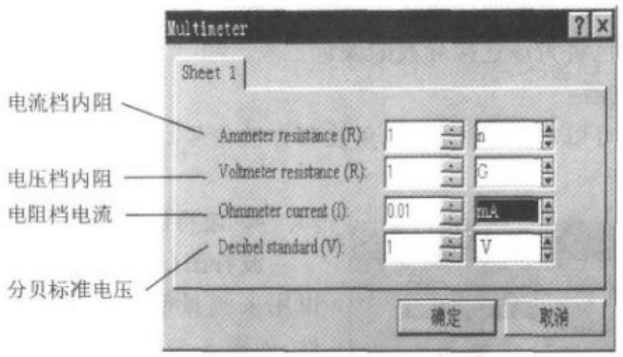


图 3.3.3　数字多用表内部参数设置对话框

图3.3.5所示。通过拖拽两指针可以详细读取波形上任一点的读数，以及两指针间读数的差。单击“Reduce”按钮可恢复原示波器面板图。单击“Reverse”按钮可改变示波器屏幕的背景颜色。单击“Save”按钮可按 ASCII 码格式存储波形读数。

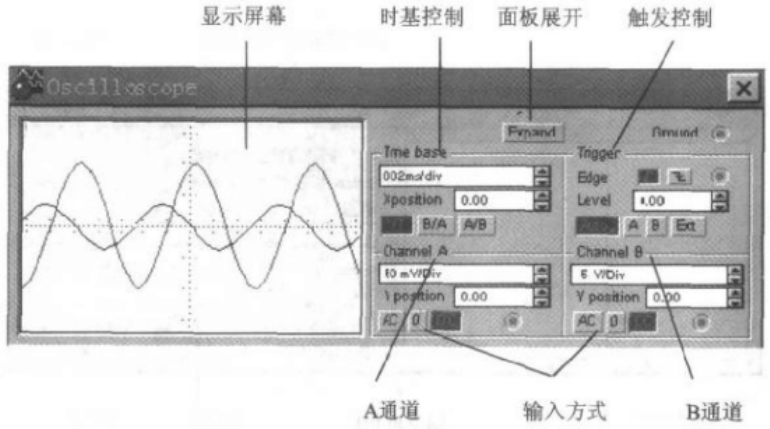


图 3.3.4　示波器

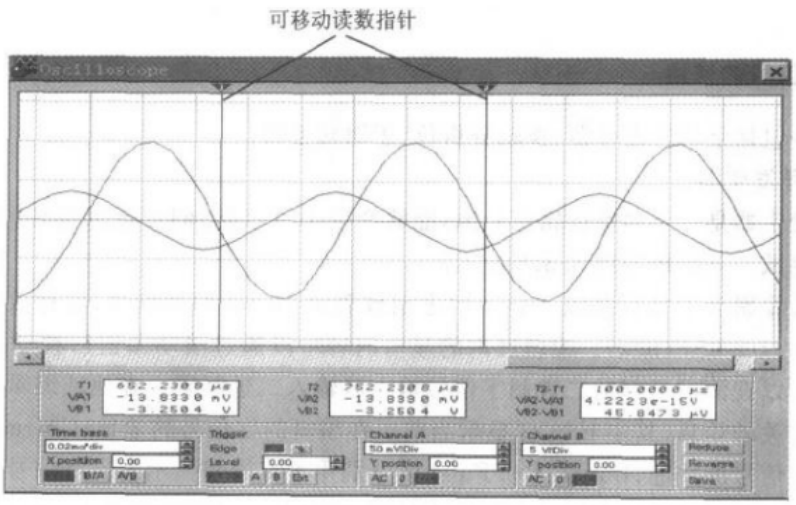


图 3.3.5　示波器面板展开

为了使示波器输出的波形便于观察测量，可设置连接示波器输入端导线的颜色，那么示波器显示该路波形的颜色将为该连接导线的颜色。

3. 函数信号发生器

函数信号发生器可以产生正弦波、三角波和方波信号。其参数设置为频率、占空比、幅度和偏置，如图 3.3.6 所示。

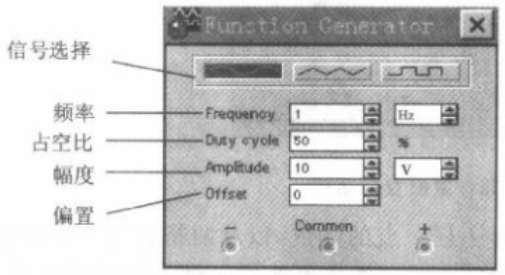


图 3.3.6 函数信号发生器

4. 波特图仪

波特图仪类似通常实验室中的扫频仪，可以用来测量和显示电路的幅频特性与相频特性，如图 3.3.7 所示。波特图仪有 IN 和 OUT 两对端口，其中 IN 端口分别接电路输入端的正、负端，OUT 端口分别接电路输出端的正、负端。在使用波特图仪时，必须在电路的输入端接入 AC（交流）信号源，其信号频率的设定无特殊要求，频率测量范围由波特图仪的参数设置决定。

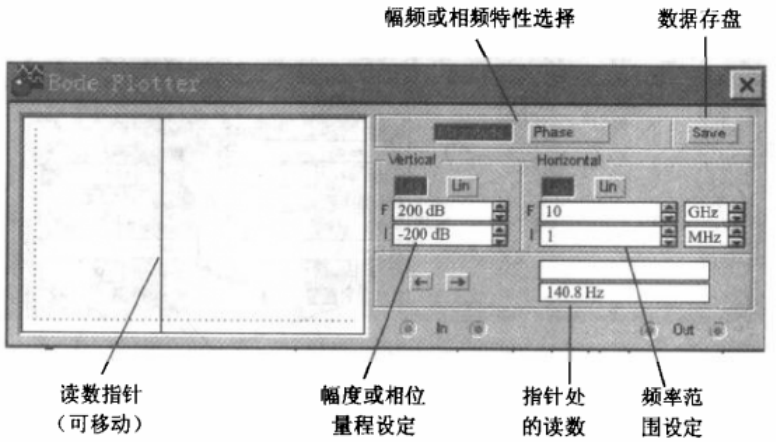


图 3.3.7 波特图仪

3.3.2 数字仪表的使用

数字仪表包括字信号发生器、逻辑分析仪、逻辑转换仪。

1. 字信号发生器

字信号发生器是一个多路逻辑信号源，能够产生 16 位（路）同步逻辑信号，用于对数字逻辑电路进行测试。其面板如图 3.3.8 所示。

在字信号编辑区中每行是以 4 位十六进制数表示 16 bit 的字信号，可存放 1 024 条字信号，地址编号为 0～3FF(hex)。编辑区内的显示内容可以通过滚动条前后移动进行十六进制数编辑。也可以在面板下部的二进制字信号输入区输入二进制码。在地址编辑区可以编辑与字信号地址有关的信息，其中 Edit 区显示当前正在编辑的字信号的地址；Current 区显示当前正在输出的字信号的地址；Initial 区和 Final 区分别用于编辑和显示输出字信号的首地址和末地址。字信号发生器按照一定规律逐行从信号输出端送出，并在每个输出端的小圆圈内实时

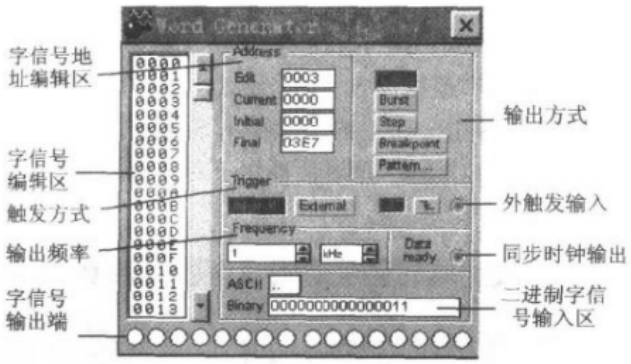


图 3.3.8　字信号发生器

显示各个位的 bit 值。

字信号的输出方式分为 Step(单步)、Burst(单帧)、Cycle(循环)三种方式。单击一次"Step"按钮,字信号输出一行,可用于对电路的单步调试。单击"Burst"或"Cycle"按钮,则字信号从首地址开始至末地址连续逐行地输出或循环不断地输出,其字信号输出节奏由输出频率的设置决定。选中某地址的字信号后,单击"Breakpoint"按钮则该地址被设置为中断点。在单帧或循环输出方式时,当运行至该地址时输出暂停,再单击"Pause"按钮或按"F9"键则恢复输出运行。

2. 逻辑分析仪

逻辑分析仪可以同步记录和显示 16 路数字信号。它可以用于对数字信号的高速采集和时序分析,是分析和设计数字系统的有力工具。其界面如图 3.3.9 所示。

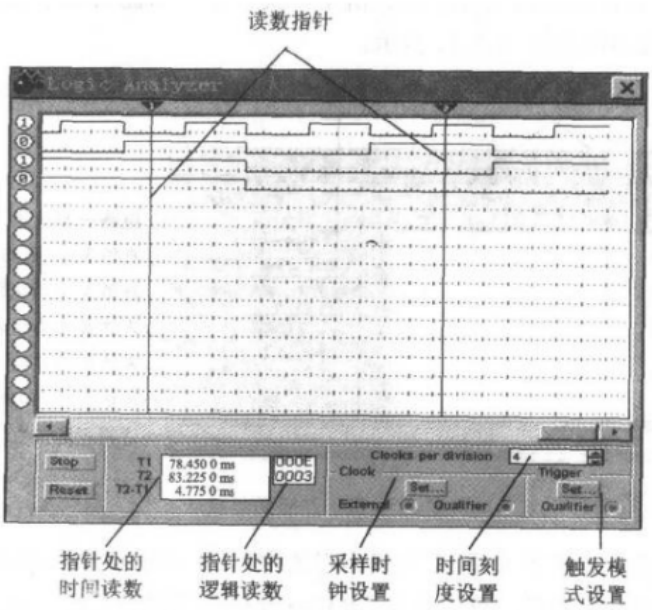


图 3.3.9　逻辑分析仪

逻辑分析仪界面左边的 16 个小圆圈对应 16 个输入端,按从上至下排列依次为最低位到最高位。每个小圆圈内实时显示各路输入逻辑信号的当前值。在波形显示区以方波形式显示

各路逻辑信号的波形，通过设置输入导线的颜色可修改相应波形的显示颜色。波形显示的时间轴刻度可通过时间刻度设置来设定。拖拽读数指针可读取波形数据，在其界面下部的方框内显示指针所处位置的时间读数和逻辑读数。

触发方式有多种选择，可以单击界面上的“Trigger/Set”按钮进行设置。通过单击“Clok/Set”按钮弹出如图 3.3.10 所示的对话框，该对话框用于对波形采集的控制时钟设置和设置触发前后数据采集的点数、触发门限值。

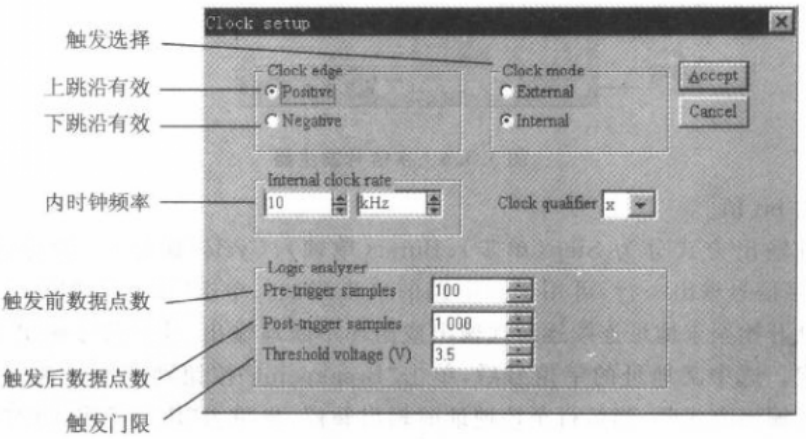


图 3.3.10　时钟控制对话框

3. 逻辑转换仪

逻辑转换仪是 EWB 特有的仪表，实际中不存在与之对应的设备。逻辑转换仪能够完成真值表、逻辑表达式和逻辑电路三者之间的相互转换。这一功能为数字逻辑电路的设计和仿真带来很大方便。其界面如图 3.3.11 所示。

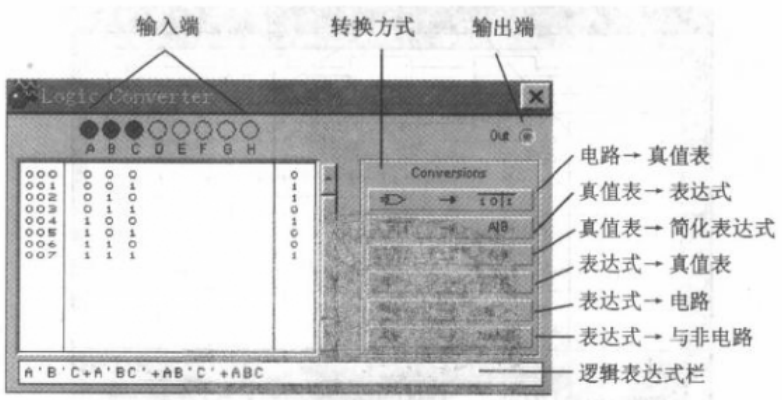


图 3.3.11　逻辑转换仪

由电路转换为真值表的方法是：首先画出逻辑电路图，将电路输入端和输出端分别连接至逻辑转换仪的输入端和输出端。然后单击“电路→真值表”按钮，立即出现该电路的真值表。

由真值表也可以导出逻辑表达式。首先根据实际输入信号的个数选定逻辑转换仪上的输入端（由 A～H）。此时真值表区自动出现输入信号的所有组合，而输出列的初值则全部为零。可以根据实际的逻辑关系修改真值表的输出值。然后单击“真值表→表达式”按钮，在仪表界

面的底部逻辑表达式栏中出现相应的逻辑表达式；如果单击“真值表→简化表达式”按钮，可得到简化的逻辑表达式。表达式中“'”表示逻辑变量的“非”。

要实现逻辑表达式转换为其他形式，直接在逻辑表达式栏中输入表达式（“与-或”式或“或-与”式均可），然后单击相应的转换方式按钮，即可得到相应的真值表或逻辑电路图。

3.4　子电路的生成与使用

为了实现较复杂的电路系统，使电路的连接简洁明了，可以将部分常用电路定义为子电路。子电路相当于用户自己定义的小型集成电路，存放在自定义元件库中可供以后反复调用。

要建立一个子电路，首先绘制好电路（不包括仪器仪表）及连接，然后单击工具栏上的生成子电路按钮或选择 Circuit/Create Subcircuit（电路/生成子电路）命令，弹出如图 3.4.1 所示的对话框，填入子电路名称并根据需要单击其中的某个按钮，这时只是完成了子电路的定义。子电路图标已经存放在 Favorites 库（自定义元件库）中。接着拖拽子电路图标至电路工作区，双击子电路图标打开子电路窗口，对它作进一步的编辑、修改和输入/输出的引出端设定。要设定和添加引出端的方法是：从子电路窗口中的某一元件拖拽引出导线至窗口的任一边沿处，待出现小方块时释放鼠标器即可。

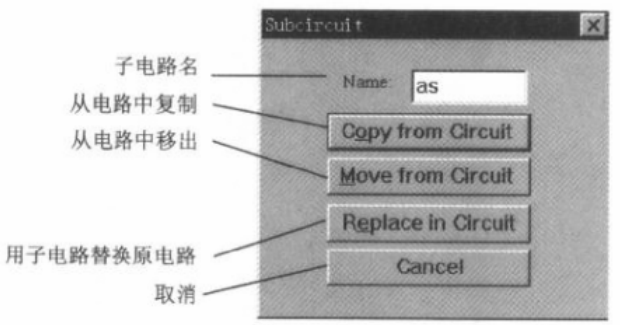


图 3.4.1　子电路设置对话框

一般情况下生成的子电路仅在本电路中有效。要应用到其他电路中，可使用剪贴板复制和粘贴操作。也可以将其粘贴到（或直接编辑）Default. EWB 电路文件的自定义元件库中。以后每次启动 EWB 软件，自定义元件库中自动包含该子电路，像使用其他元件一样可反复调用子电路。

3.5　网表文件转换和印制线路板设计

网表文件是一种采用文本格式描述电路的文件，EWB 可以同 SPICE 的网表文件相互转换，即可以把在工作区内创建的电路以 SPICE 的网表文件形式输出，也可以输入 SPICE 的网表文件转换成电路图。同时 EWB 还可以把电路图转换成各种印制线路板排版软件能接受的网表文件。它与常用的电路设计和排版软件如 ORCAD PCB（*. NET），Tango（*. NET），Eagle（*. SCR），Protel（*. NET），Layout（*. PLC）等软件相衔接，直接排出相应的印制线路板。其转换的步骤为如下：

① 创建电路，完成电路的分析和设计工作，然后除去排版不需要的测试仪器和多余的元器件。

② 对电路中的元器件和结点进行标识，便于线路板中的识别。

③ 删除 EWB 软件和排版软件之间不相互支持的部分特殊器件。

④ 选择“文件”菜单中的“输出()”项，根据对话框要求，将电路转换为相应排版软件的网表文件。

⑤ 启动 PCB 排版软件，从“网表文件输入”项输入 EWB 软件生成的电路网表文件，然后进行印制线路板的排版工作。

第四章　EWB的元器件库

电子工作台为用户提供了非常丰富的元器件库及各种常用测试仪器，根据元器件不同类型被分为：信号源库(Source)、基本元件库(Basic)、二极管(Diode)与晶体管(Transistors)库、模拟(Analog)与数字(Digital)集成电路库、混合集成电路库(Mixed ICs)、逻辑门(Logic Gates)与数字器件(Digital)库、指示部件库(Indicators)、控制部件库(Controls)、其他器件库(Miscellaneous)和自定义库(Favorites)。这些库都以图标形式显示在电子工作台的基本操作界面上。

在建立和编辑电路图时，只要单击所需元器件库的图标，就会显示该库中所有元器件的图形，利用鼠标把所要元件拖拽至电路工作区内，可以放置在该区中任意地方。若要调整所选元器件原来的缺省设置参数，只需用鼠标双击该元器件或者单击工具栏中的器件特性按钮，就会弹出该元器件特性对话框，主要打开(模型)或者(数值)选项，根据显示该元件的参数设置内容进行修改和设定，详见附录一。可以通过按 F1 键或选择帮助命令，了解所选元件的含义、性能和使用方法。

4.1　信号源与基本元件库

如图 4.1.1 所示，EWB 提供了全部的独立电源、受控源及时钟、调幅调频信号等各种信号源。图 4.1.2 为基本元件库，主要包括常用的无源元件。

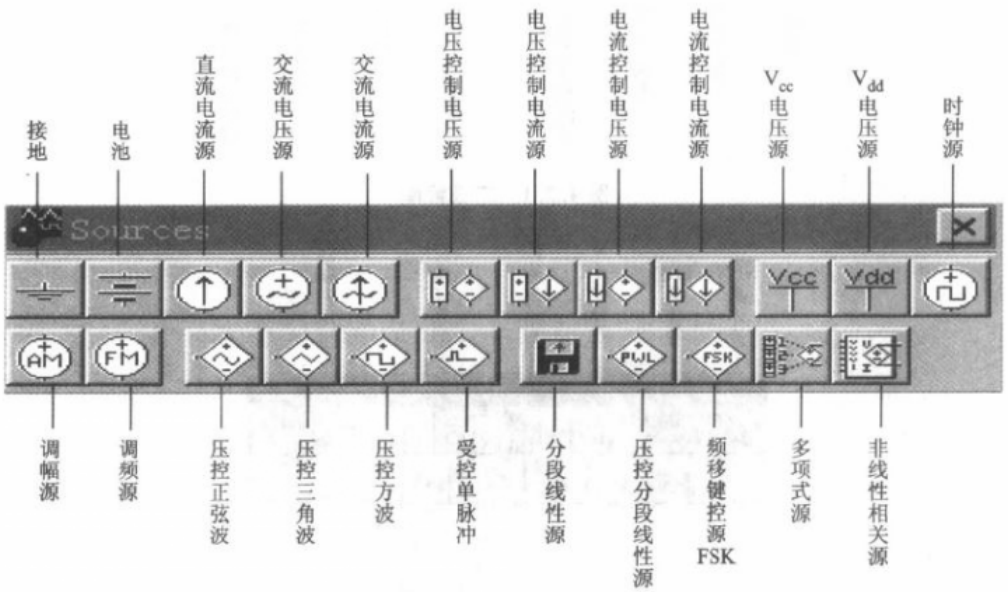


图 4.1.1　信号源

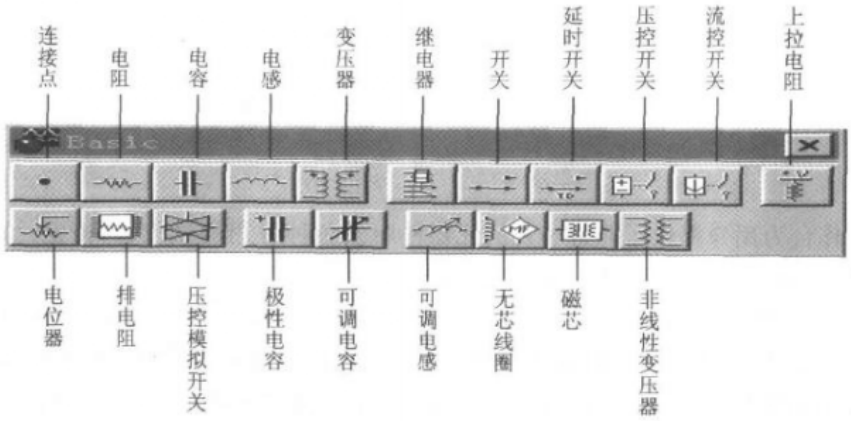


图 4.1.2 基本元器件库

4.2 二极管与晶体管库

EWB 提供的二极管元件，包括一般二极管、齐纳二极管、发光二极管、整流电路及可控硅等。晶体管库主要为双极型晶体管（三极管）、结型及 MOS 场效应晶体管，如图 4.2.1 和图 4.2.2 所示。

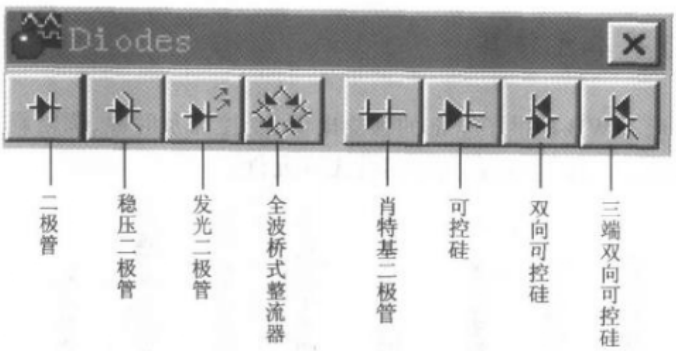


图 4.2.1 二极管库

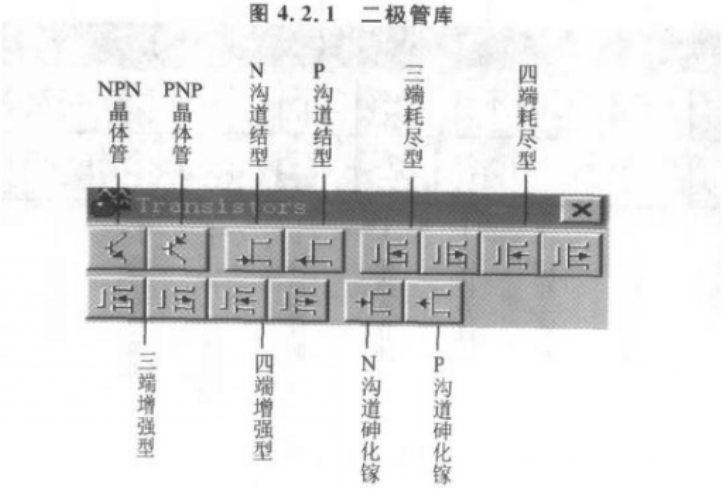


图 4.2.2 晶体管库

4.3　模拟与数字集成电路库

模拟集成电路库主要是各类放大器、比较器及锁相环电路等。数字集成电路库提供了74系列和4X系列集成电路，如图4.3.1所示。

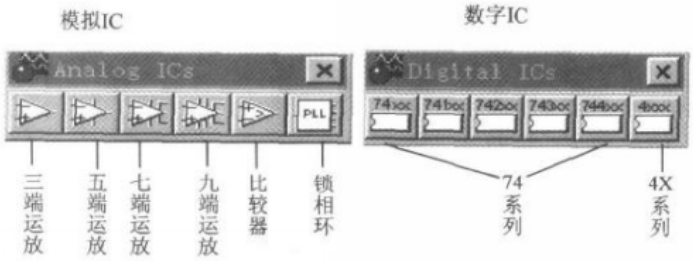


图4.3.1　模拟IC与数字IC

4.4　混合集成电路库

混合集成电路库如图4.4.1所示。

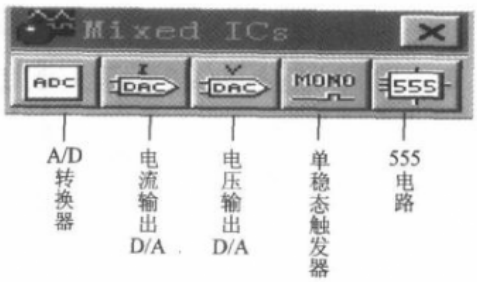


图4.4.1　混合集成电路库

4.5　逻辑门与数字器件库

逻辑门是指小规模的基本数字门，如与门、或门、非门、异或门等，如图4.5.1所示。数字器件包括具有一定功能的单元集成电路，如全加器、触发器、选择器、编码器、计数器及移位寄存器等，如图4.5.2所示。

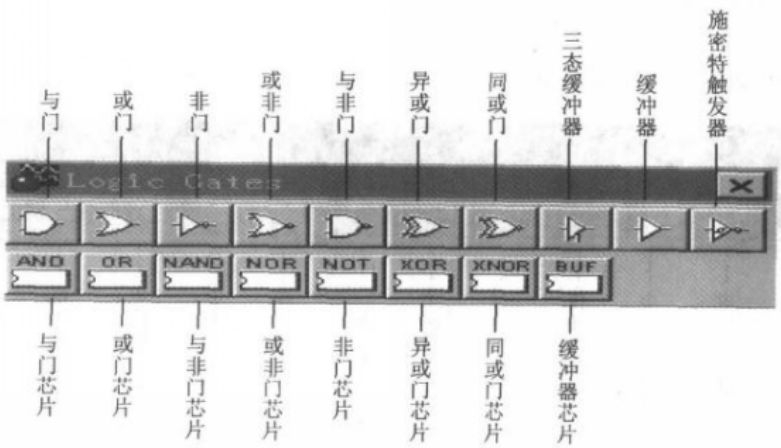


图4.5.1　逻辑门电路库

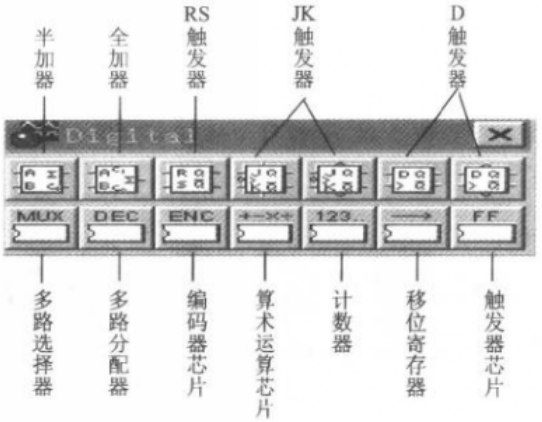


图 4.5.2 数字器件库

4.6 指示部件库

指示部件库如图 4.6.1 所示。

图 4.6.1 指示部件库

4.7 控制部件库

控制部件库如图 4.7.1 所示。

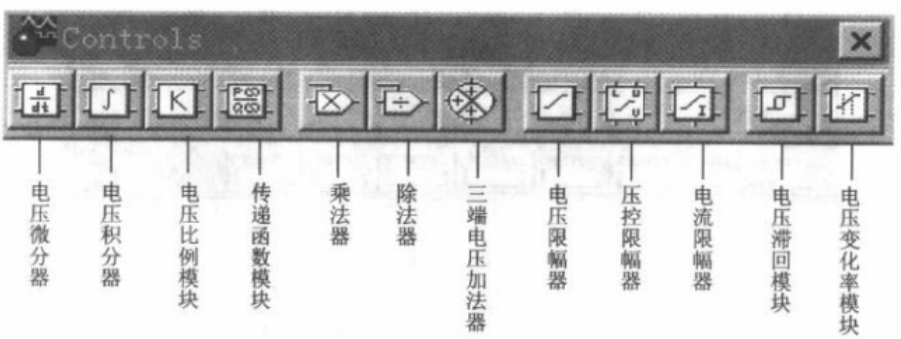


图 4.7.1 控制部件库

4.8　其他器件库

其他器件库如图 4.8.1 所示。

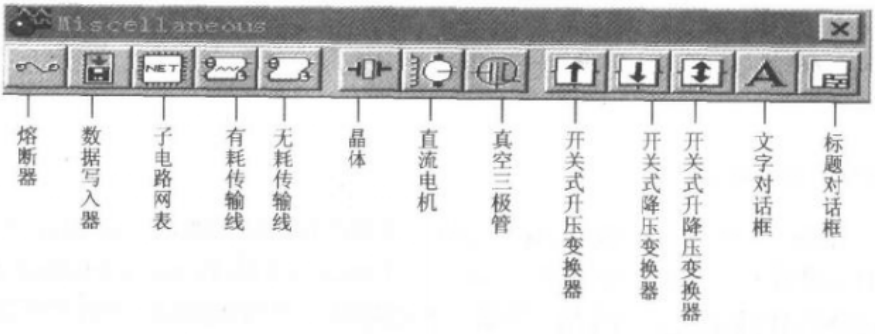


图 4.8.1　其他器件库

4.9　元器件库及元器件的创建和删除

EWB 软件已经包含了电子电路常用的元器件，但对于特殊元器件或者非常用元器件，没有包括在元器件库内。不过可以采用自建元器件库和元器件的方法。

自建元器件库和元器件主要是针对一些含有复杂器件的模拟单元电路。其方法可以分为两种：一种是采用创建“子电路”的方法，组合成一个“电路模块”，存入在自定元件库中，可反复直接调用；另一种方法是采用库中已有元器件的模型，修改元器件内部参数，建立特殊的元器件。其操作步骤如下。

1. 创建元器件名

① 双击被选元器件，会显示该元器件特性对话框。

② 单击“New Library”选项。

③ 在新库名中填入创建的元器件名，然后单击“OK”按钮，即完成元器件名的创建。

2. 编辑元器件参数

① 先选择缺省元器件库(Default)中的理想(Ideal)模型，或者其他已在库中的元器件模型，执行复制(Copy)命令。

② 选中新的元器件名，写入自己设定的新模型名。再执行粘贴(Paste)命令，这时新模型中的元器件参数为原来模型中的内容。

③ 选中新模型名，执行编辑(Edit)命令，对复制模型中的参数进行修改，然后单击“OK”按钮即完成新元器件的建立。

对于在工作区内的元器件，可以按“Delete”键将它删除。对于在元器件库中的元器件，先选中需要删除的元器件，选择对话框中的“Delete”选项即可将其删除。若想删除元器件的库名，则需打开 Windows 的文件管理器，找到 EWB 元器件库的子目录名“Models”中需要删除的库名，并将其删除。

第五章　EWB的电路仿真及分析

5.1　仿真的基本原理及参数设置

5.1.1　仿真的基本原理

电子工作台(EWB)可以对模拟、数字或混合电路进行电路的性能仿真和分析，其分析方法和元器件库的模型均建筑在SPICE(Simulation Program with Integrated Circuit Emphasis)程序(SPICE3F5)的基础上。当使用者绘制一个线路图，并按下启动开关后，就可以从测试仪器上观察输出波形和被测数据。而实际过程是该软件通过计算所创建的电路数学表达式，求得数值解。在电路中的每个元器件都有其设定的数学模型，所以这些器件模型的精度决定电路仿真结果的精度。

采用EWB软件是以SPICE3F5为模拟的核心算法，对电子线路进行仿真运行。在处理直流线性电路时，是将它作为一般非线性电路的一个特例来进行处理的。在求解电路方程或在求解系统节点方程时，将状态方程的A矩阵分成两个三角矩阵：下三角矩阵L和上三角矩阵U，并采用前向迭代法和后向迭代法求解这两个矩阵的解。为了更快、更有效地求出电路节点的数值解，提高计算精度，采用两种不同的算法，即

① 局部枢轴算法(A Partial Pivot Algorithm)：它减少由于“LU”分解方式带来的截断误差。

② 预定阶算法(A Preordering Algorithm)：它改善矩阵的稳定条件，减少方程求解的非零项。

为了求解一般非线性电路中的直流工作点，EWB软件采用了两种改良的牛顿-拉夫申算法(Newton - Raphson Algorithm)——“Gmin stepping”算法和“Source stepping”算法。前一种算法是一个多种步长的迭代算法。后一种算法是一种辅助收敛算法，在求解非线性电路方程时，该算法设置一个源矢量因子，以加速直流解的收敛。它把单步长转换为多种步长的非线性电路方程来求解。这两种算法的步长可以通过菜单命令Analysis\Analysis Options进行设定。

5.1.2　分析方法的参数设置

EWB软件可以根据用户对电路分析的要求，对电路仿真的总体参数进行设置。因此仿真效果与使用者如何设置菜单命令Analysis\Analysis Options中的参数有很大关系。下面介绍“分析任选项”(Analysis Options)中参数的含义和如何进行设置。

首先选择菜单命令Analysis\Analysis Options，屏幕弹出一个对话框。它包括五个选项：“Global”(总体分析选择)、“DC”(直流分析选择)、“Transient”(瞬态分析选择)、“Device”(器件分析选择)、“Instruments”(仪器分析选择)。然后用户根据需要对其选项中的参数进行调

整和设置。下面列表依次说明这些选项中的参数的含义和设置要求。

1. 总体分析选择

总体(Global)分析选择如表 5.1.1 所列。

表 5.1.1　总体分析选择

总体分析选项	含义和设置要求
ABSTOL	电流的绝对精度，通常小于电路中最大电流信号的 6～8 个数量级。缺省设置：1.0e－12，适合一般双极型晶体管、VLSI 电路
GMIN	最小电导。该值不能设置为零。缺省设置：1.0e－12。增大该值有利于收敛性，但会影响仿真的精度。一般不调整
PIVREL	最大矩阵项与主元值的相对比率。缺省设置：0.001，一般不需修改
PIVTOL	主元矩阵项绝对最小值。缺省设置：1.0E－13。一般情况不需修改
RELTOL	相对误差精度。改变此值会影响仿真速度和收敛性。取值范围在 0.01～1.0e－06 之间，缺省设置：0.001
TEMP	仿真温度。缺省设置：27 ℃
VNTOL	电压绝对精度。通常小于电路中最大电压信号的 6～8 个数量级。缺省设置：1.0e－6
CHGTOL	电荷绝对精度。缺省设置：1.0e－14
RAMPTIME	斜升时间。在确定时间内独立源、电容和电感从零至终值的条件
CONVSTEP	相对收敛步长限制。在求解直流工作点时，建立相对步长限制自动控制收敛。缺省设置：0.25
CONVAB－SSTEP	绝对收敛步长限制。在求解直流工作点时，建立绝对步长限制自动控制收敛。缺省设置：0.1
CONVLIMIT	收敛限制。用于某些元件模型内部的收敛算法。缺省设置：选用(ON)
RSHUNT	模拟节点分流电阻。在节点和地之间接入电阻，该值应该较大。缺省设置：不选用。若选用则该值为 1.0e－12
临时文件大小	仿真时临时性文件规模。当储存仿真结果的文件达到设定的最大容量时，会出现对话框：停止仿真、使用剩余磁盘空间、删除已有数据继续仿真。缺省设置：10 MB

2. 直流分析选择

直流(DC)分析选择如表 5.1.2 所列。

表 5.1.2　直流分析选择

直流分析选项	含义和设置要求
ITL1	工作点分析迭代极限。限制 N－R 算法的迭代次数。缺省设置：100。若出现“直流分析时不收敛”等情况，则可增加该值从 500～1 000
GMINSTEPS	GMIN 算法的步长。缺省设置：10
SRCSTEPS	SOURCE 算法的步长。缺省设置：10

3. 瞬态分析选择

瞬态(Transient)分析选择如表 5.1.3 所列。

表 5.1.3 瞬态分析选择

瞬态分析选项	含义和设置要求
ITL4	瞬态分析每时间点迭代次数的上限。增大此值会缩短瞬态分析的时间,但过分降低该值会引起不稳定。缺省设置:10。若出现"时间步长太小"或"瞬态分析不收敛",则可增大此值为 15～20
MAXORD	积分方法的最大阶数。一般采用缺省设置:2,取值范围:2～6
TRTOL	瞬态误差精度因素。缺省设置:7。一般情况不需要修改
METHOD	瞬态分析数字积分方法。缺省设置:Trapezoidal(梯形法)适合振荡电路模式,Gear(变阶积分)适合其他电路,例如有理想开关的电路
ACCT	打印数据。显示仿真过程的有关信息。缺省设置:OFF

4. 器件分析选择

器件(Device)分析选择如表 5.1.4 所列。

表 5.1.4 器件分析选择

器件分析选项	含义和设置要求
DEFAD	MOSFET 漏极扩散区面积。缺省设置:0
DEFAS	MOSFET 源极扩散区面积。缺省设置:0
DEFL	MOSFET 沟道长度。缺省设置:0.000 1
DEFW	MOSFET 沟道宽度。缺省设置:0.000 1
TNOM	模型参数标称温度。缺省设置:27 ℃。一般不要调整
BYPASS	非线性模型评估器件。缺省设置:ON。若选 OFF 将增加仿真时间
TRYTOCOMPACT	小型传输线数据。只用于有耗传输线的访真。缺省设置:OFF

5. 仪器分析选择

如图 5.1.1 所示为仪器(Instruments)分析选择界面。仪器分析选择主要包括示波器显示方式、初始条件、波特图仪的显示点数等参数的设置。

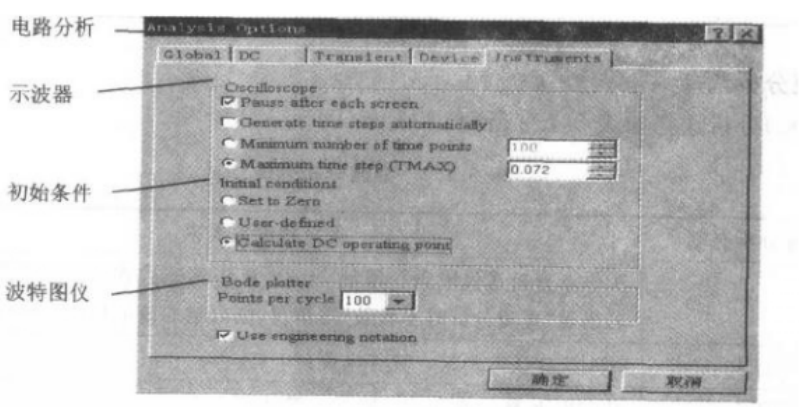


图 5.1.1 仪器分析选择界面

5.2　电路的基本分析方法

EWB提供了多种电路分析方法和手段，可以完成电路的基本或常规分析，如直流参数、交流参数、瞬态参数以及噪声分析、失真分析和傅里叶分析。这些常规参数分析为设计人员分析和设计电路提供了很大帮助。

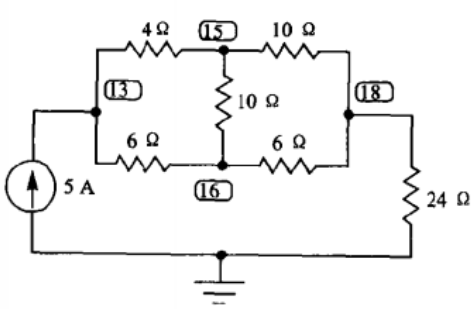
5.2.1　直流工作点分析

直流工作点(DC Operating Point)或静态工作点分析的目的是确定电路的工作点，并且是对电路的其他性能进一步分析的基础。比如在对晶体管进行交流分析时，首先需要计算它的直流工作点。在进行直流分析时，将电路中的交流(AC)源设为零、电容开路、电感短路。另外，在直流分析时可将数字元器件视为大电阻接地。以下为直流工作点分析步骤：

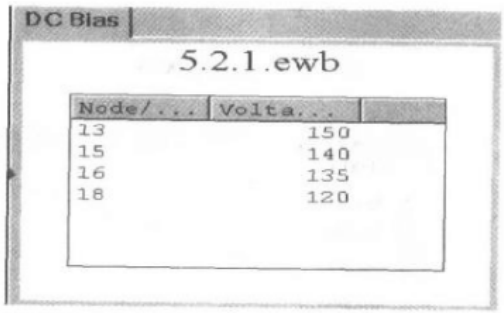
① 创建好需要进行分析的电路图，选择菜单命令 Circuit\Schematic Options，选定 Show Nodes(显示节点)，把电路的节点标志(ID)显示在电路图上。

② 选择菜单命令 Analysis(分析)\DC Operating Point，会弹出一个显示窗口(Display Graph)，在窗口中会自动把电路中所有节点电压和电源支路电流的数值显示出来。

举例分析：分析电路如图 5.2.1(a)所示，直流工作点分析结果如图 5.2.1(b)所示。



(a)



(b)

图 5.2.1　直流工作点分析电路及结果

5.2.2　交流频率分析

交流频率(AC Frequency)分析主要分析输出与输入关系相对于频率变化的响应。需先选定被分析的电路节点，在分析时电路中的直流源将自动置零，交流电源、电容、电感等均处在交流模式。电路的输入信号将被忽略，会自动由内部的正弦波信号替代输入。因此输出响应就是该电路随交流频率的函数。其操作过程如下：

① 选择菜单命令 Analysis\AC Frequency，弹出 AC Frequency Analysis 对话框。

② 在对话框中，确定需要分析的电路节点、起始频率、终点频率、扫描形式(十进制/线性/倍频程)、显示点数和纵向尺度(线性/对数/分贝)。其对话框的界面如图 5.2.2 所示。

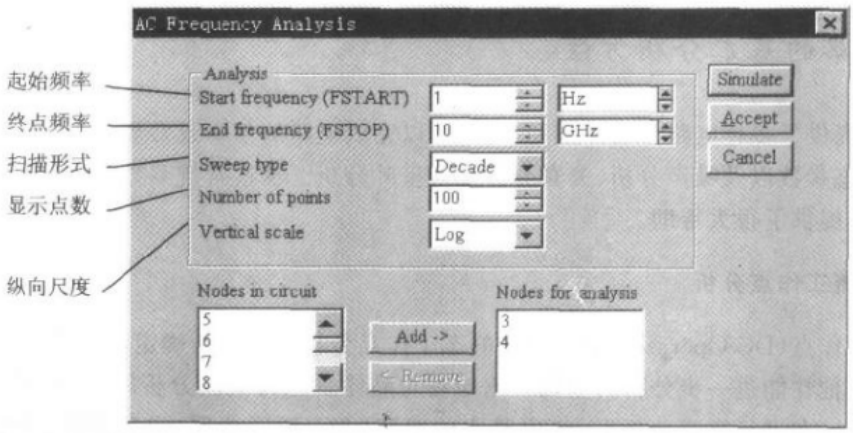


图 5.2.2 交流频率分析参数设置

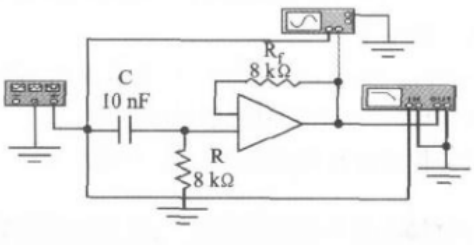


图 5.2.3 高通滤波器

③ 单击对话框中的“Simulate(仿真)”按钮即开始分析，并弹出频率特性波形的显示窗口。

举例分析：对如图 5.2.3 所示的高通滤波器经交流频率分析后，得到如图 5.2.4 所示的交流频率响应图。如果把波特图仪连接到该电路，按启动开关进行仿真，则可得到类似的分析。

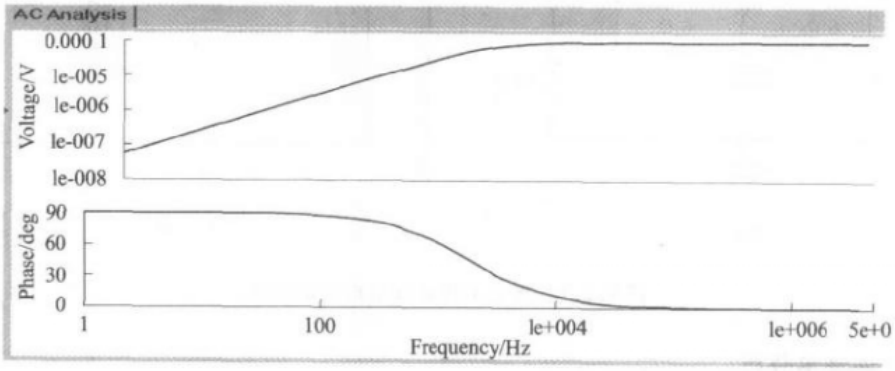


图 5.2.4 高通滤波器交流频率响应图

5.2.3 瞬态分析

瞬态(Transient)/暂态分析是指对所选定的电路节点的时域响应。即观察该节点在整个显示周期中每一时刻的电压波形。在分析时，直流电源保持常数，交流信号源随着时间改变，电容和电感都是能量储存模式元件。

在对选定的节点作瞬态分析时，一般可先对该节点作“直流工作点分析”，这样直流分析的结果可作为瞬态分析的初始条件。其瞬态分析步骤如下：

① 先确定要分析的节点，选择菜单命令 Analysis\Transient，弹出瞬态分析对话框。

② 在对话框中进行参数设置，如表 5.2.1 所列。

③ 单击对话框中的“Simulate”按钮（如按“Esc”键将停止仿真运行）。

表 5.2.1　瞬态分析对话框参数设置

瞬态分析	含义和设置要求
Set to zero	初始条件设为零。缺省设置：不选
User - defined	用户定义初始条件。缺省设置：不选
Calculate DC operating point	计算直流工作点作为初始条件。缺省设置：选用
Start time	进行分析的起始时间。缺省设置：0 s
Stop time	进行分析的终点时间。缺省设置：0.001 s
Generate time - steps automatically	自动选择一个较为合理的或最大的时间步长。缺省设置：选用
Minimum - number - of points	仿真输出的点数。缺省设置：100 点
Maximum time step	仿真时能达到的最大时间步长。缺省设置：1e-05 s
Nodes for analysis	被分析的节点

举例分析：如图 5.2.5(a)所示为一个 RC 电路，当输入电压 5 V 时，电容两端经过充电过程最终达到 5 V。其瞬态分析波形如图 5.2.5(b)所示。

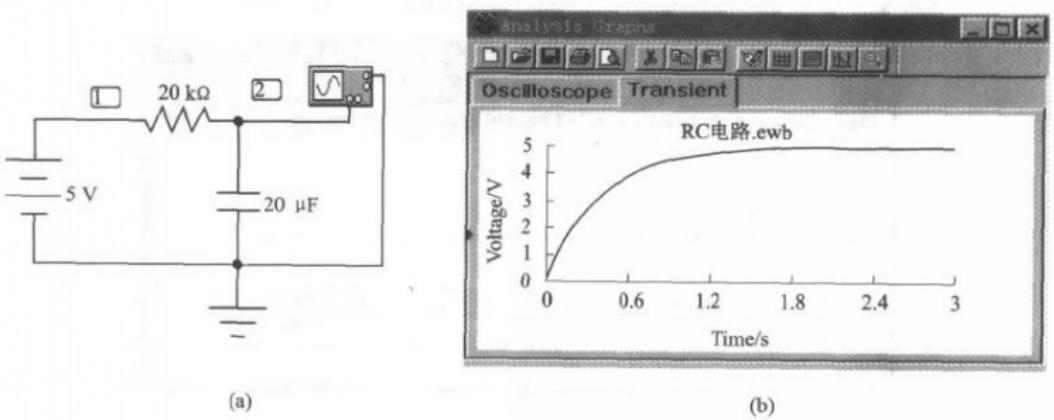


图 5.2.5　RC 电路及瞬态分析

瞬态分析的结果，即为电路中该节点的电压波形图。也可以使用示波器，把它连到需分析的节点上，打开启动按钮，得到相同的结果。但采用瞬态分析方法，可以通过设置，更仔细地观察到波形起始部分的变化情况。

5.2.4　傅里叶分析

傅里叶(Fourier)分析方法用于分析时域信号的直流分量、基波分量和谐波分量。对被测节点的时域信号进行离散傅里叶变换，求出它的频域变化规律。在进行分析时，首先选择被分析的节点，一般将电路中所用信号源的频率作为基频；若在电路中有多个信号源时，则可将基

频设定为各交流信号源的最小公因数。例如有一个 10.5 kHz 和一个 7 kHz 信号源，则基频设定为 0.5 kHz。其分析步骤如下：

① 选择菜单命令 Analysis\Fourier，弹出傅里叶分析对话框。

② 根据对话框要求，设置参数。其对话框中参数的含义和设置要求见表 5.2.2。

③ 单击对话框中的“Simulate”按钮（如按“Esc”键将停止仿真运行）。

表 5.2.2 傅里叶分析对话框的参数设置

傅里叶分析	含义和设置要求
Output variable	输出变量、被分析的节点
Fundamental	傅里叶分析的谐波基频，为交流源的频率或最小公因数。缺省设置：1 Hz
Number of har - monics	被计算、显示的基频谐波数。缺省设置：9
Vertical scale	纵向尺度，线性/对数/分贝。缺省设置：线性
Display phase	显示相频特性曲线。缺省设置：不选
Output as line	显示幅频特性曲线，不是傅里叶变换的条形图。缺省设置：不选

傅里叶分析可以显示被分析节点的电压幅频特性和相频特性，显示的幅度可以是离散条形也可以是连续曲线型，缺省设置为离散型。例如：分析如图 5.2.5(a)所示的 RC 电路，可求得节点(2)的傅里叶变换波形，取基频为 1 kHz，显示的离散傅里叶图形如图 5.2.6 所示。

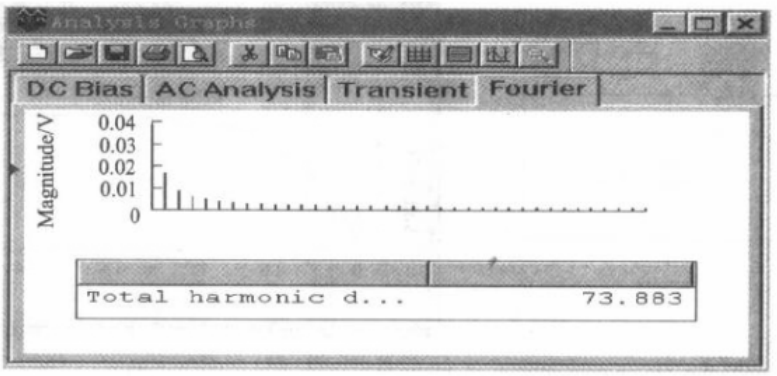


图 5.2.6 离散傅里叶变换图形

5.2.5 噪声分析

噪声(Noise)分析用来检测电子电路输出端噪声源的大小，用于计算、分析电路中由电阻及半导体元件所产生的噪声总和。在分析时，假设电路中各噪声源互不相关，因此它们的数值可以分开独立计算。总噪声是每个噪声源对于特定的输出节点产生的噪声均方根值的和。举例来说，在噪声对话框中选择 V1 为“输入噪声参考源(Input noise reference source)”，把 N1 作为“输出节点(Output node)”，则电路中各噪声源在 N1 处形成的输出噪声，等于把该值除以 V1～N1 的增益获得的等效输入噪声，再把这个等效输入噪声输入到一个无噪声的电路中，即先前计算的输出噪声值将出现在 N1 处。其分析步骤如下：

① 选择 Analysis\Noise 菜单命令，弹出一个对话框。其界面如图 5.2.7 所示。

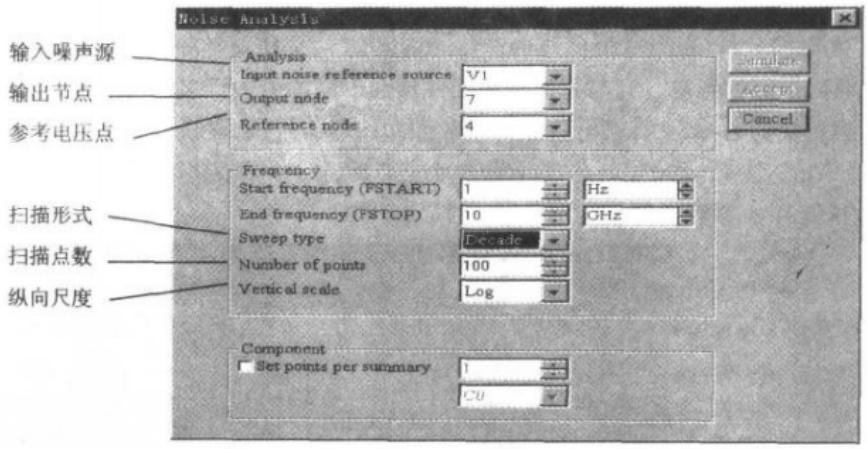


图 5.2.7　噪声分析参数设置对话框

② 根据对话框的要求，设置参数。

③ 单击对话框中的“Simulate”按钮进行分析（如按“Esc”键将停止仿真运行）。

该分析方法主要是用于小信号电路的噪声分析，元器件噪声模型采用 SPICE 模型。噪声分析完成后，在显示窗口中显示的是输出噪声功率谱和输入噪声功率谱，单位为 V/Hz。

5.2.6　失真分析

信号的失真通常是由电路的增益非线性和相位不一致所造成的。非线性增益造成谐波失真；而相位的不一致造成互调失真（调制失真）。若电路中有一个交流信号源，则该分析方法能确定电路中每一个节点的二次谐波和三次谐波的复值。若电路中有两个交流信号源，则该分析方法能确定电路节点在三个不同频率处的复值，即两个频率之和的值、两个频率之差的值以及二倍频与另一个频率之差的值。

该分析方法是对电路进行小信号的失真分析，采用多维的“Volterra”分析法和多维“泰勒（Taylor）”级数来描述工作点处的非线性，级数要用到三次方项。这种分析方法尤其适合观察在瞬态分析中无法看到的、比较小的失真。其分析方法如下：

① 选择 Analysis\Distortion 菜单命令，弹出一个对话框。

② 根据对话框的要求，设置参数。

③ 单击对话框中的“Simulate”按钮进行分析（如按“Esc”键将停止仿真运行）。

该分析方法主要被用于小信号模拟电路的谐波失真和内部调制失真的分析，元器件采用 SPICE 模型。

5.3　电路特性的高级分析方法

EWB 不仅提供了以上常规的分析方法即常规参数的分析，而且还提供了几种高级分析，如参数扫描分析、温度扫描分析、零-极点分析、传递函数分析、直流和交流灵敏度分析、蒙特卡罗分析等。

5.3.1 参数扫描分析

采用参数扫描分析(Parameter Sweep Analysis)方法分析电路,可以较快地获得某个元件的参数变化时对电路的影响。它与每次利用元件参数不同的值进行多次仿真的结果相同。设计者可通过参数扫描对话框,选择被分析元件参数的初始值、终值和增量值。其分析步骤如下:

① 选择 Analysis\Parameter Sweep 菜单命令,弹出一个对话框,如图 5.3.1 所示。确定并输入要分析元件的参数及分析的节点。

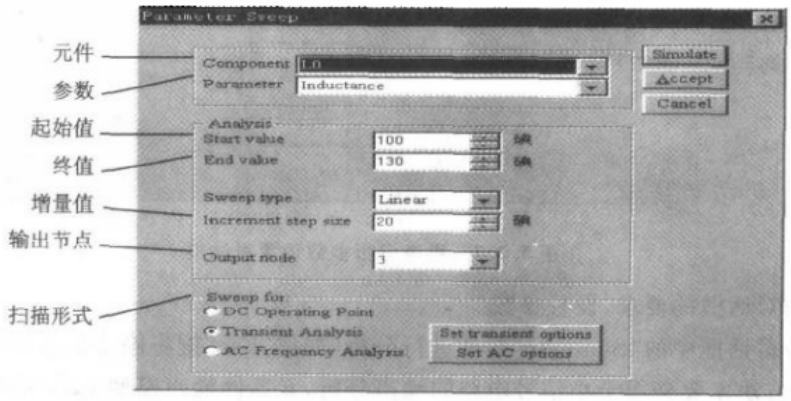


图 5.3.1 参数扫描分析参数设置对话框

② 选择扫描形式:DC 工作点、瞬态或 AC 频率分析。

③ 单击对话框中的"Simulate"按钮进行分析(如按"Esc"键将停止仿真运行)。

在参数扫描对话框的扫描形式选项中,若选择瞬态分析或交流分析时,则可以单击"Set transient options"或"Set AC options"按钮,将打开另一个对话框,可以观察和修改其他参数。

参数扫描分析是根据元件不同取值绘出对应的多种输出曲线。若采用线性(Linear)扫描方式时,则扫描参数曲线数等于终值减去初始值除以增量值。增量值的设置仅适合线性扫描情况。如果采用十进制(Decade)扫描方式时,则曲线数等于初始值乘以 10 倍数的递增,直到终值。如果采用八进制(Octave)扫描方式时,则曲线数等于初始值乘以 2 倍数的递增,直到终值。对于数字器件,在分析时被视为高阻接地。

举例分析:晶体管振荡电路如图 5.3.2(a)所示,分析的参数为电路中的电感 L,观察电感参数变化对电路振荡频率的影响。其测试结果如图 5.3.2(b)所示。

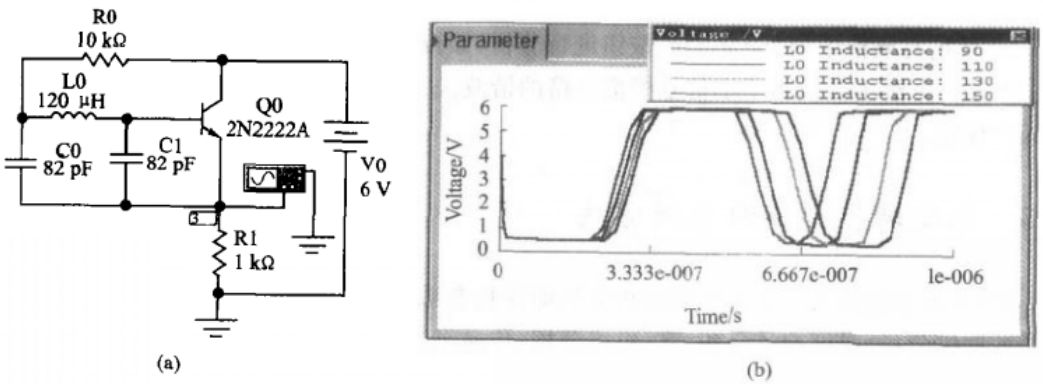


图 5.3.2 晶体管振荡电路及振荡波形

5.3.2　温度扫描分析

采用温度扫描(Temperature Sweep)分析可以同时观察到在不同温度条件下的电路特性,相当于该元件每次取不同的温度值进行多次仿真。在进行其他分析时,电路的仿真温度缺省设置为 27 ℃。其操作步骤如下:

① 选择 Analysis\Temperature Sweep 菜单命令,弹出一个对话框,确定要分析的元件和节点。

② 根据对话框要求,设置参数(与参数扫描分析对话框的参数设置相似)。

③ 单击对话框中的"Simulate"按钮进行分析(如按"Esc"键将停止仿真运行)。

温度扫描分析输出的是各种不同温度值的曲线,可采用线性、十进制或者倍频等时间扫描方式。其显示的曲线窗口同参数扫描分析时完全相同。对于数字器件,视为高阻接地。

5.3.3　零-极点分析

零-极点(Pole - Zero)分析可决定一个电路的小信号交流传递函数的零点和极点。通常先进行直流工作点分析和计算,得出非线性元件的线性化的小信号模型。在此基础上再分析传递函数的零、极点。根据传递函数(转移函数)输出电压、电流和输入电压、电流之间的关系,得到电压传递函数、电流传递函数、转移阻抗、输入阻抗、输出阻抗等等。零-极点分析方法采用 SPICE 算法,可能会产生错误的信息,如"零-极点分析达到重复计算 200 次极限后将停止",但仍可找到所有的零点和极点。其执行过程如下:

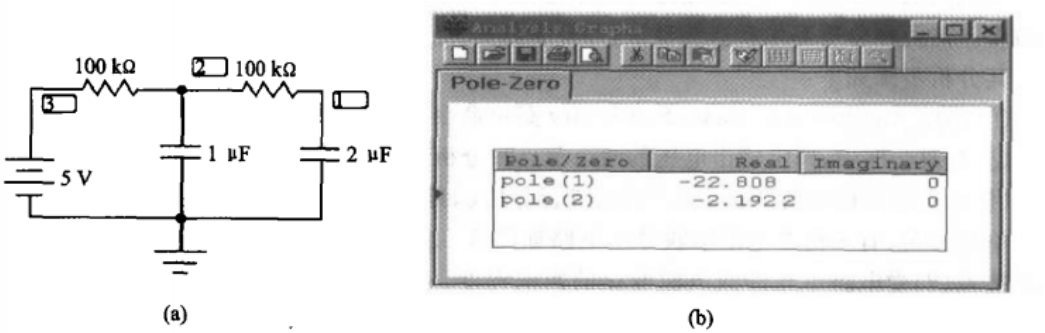
① 选择 Analysis\Pole - Zero 菜单命令,弹出一个对话框。

② 在对话框中,确定输入节点(Input+和 Input−)及输出节点(Output+和 Output−),可以使用接地点为输入/输出节点的参考点,并且还要选择分析类型(零点和极点分析)。

③ 单击对话框中的"Simulate"按钮进行分析(如按"Esc"键将停止仿真运行)。

零-极点分析主要用于模拟小信号电路的分析,对于数字器件,视为高阻接地。

举例分析:二阶 RC 电路如图 5.3.3(a)所示,求得节点(1)对输入电压(节点(3))的电压传递函数的零-极点,如图 5.3.3(b)所示。



(a)　　(b)

图 5.3.3　二阶 RC 电路及零-极点

5.3.4　传递函数分析

传递函数(Transfer Function)分析是计算电路中指定的两输出节点与输入信号源间交流

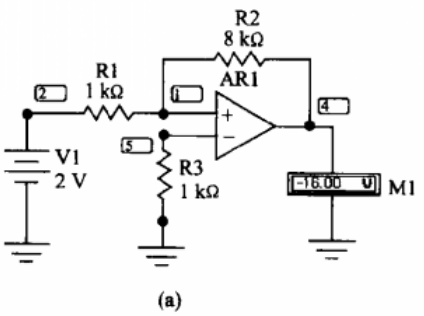
小信号的传递函数(转移函数),也可计算电路的输入和输出阻抗。需先对模拟电路或非线性器件进行直流工作点分析,求得线性化的模型,再进行小信号传递函数分析。输出变量可以是电路中的节点电压,输入必须是独立源。其分析步骤如下:

① 选择 Analysis\Pole - Zero 菜单命令,弹出一个对话框。

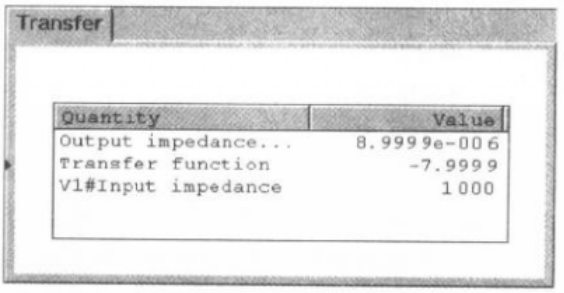
② 根据对话框要求,设置参数(它与参数扫描分析对话框的参数设置相似)。

③ 单击对话框中的"Simulate"按钮进行分析(如按"Esc"键将停止仿真运行)。

分析结果弹出一窗口显示传递函数增益、输入和输出阻抗的值。由运算放大器组成反相比例放大电路,如图 5.3.4(a)所示。其分析结果如图 5.3.4(b)所示。



(a)



(b)

图 5.3.4 运算放大器及传递函数分析结果

5.3.5 直流和交流灵敏度分析

灵敏度(Sensitivity)是指电路中任一节点的电压或支路电流对电路中元件参数的敏感程度。对于网络函数 T(x)(x 为其中某一元件的参数),可以定义函数 T(x)对参数 x 的灵敏度 S_x^T,即

$$S_x^T = dT/dx$$

该分析方法可以计算某节点电压和支路电流的交、直流灵敏度。直流灵敏度的分析是在确定直流工作点的基础上,计算电路中所有元件参数变化引起指定电压或电流发生变化的程度。交流灵敏度的分析是计算指定电压或电流对于电路中一个元件参数的交流小信号状态的灵敏度。这两种分析都用参数扰动法,计算参数变化对电压或电流的影响,并列表显示计算结果。其分析步骤如下:

① 选择 Analysis\DC &AC Sensitivity 菜单命令,弹出一个对话框。

② 在对话框中,确定输出电压的两个节点和分析方式(DC 或 AC 灵敏度)。

③ 单击对话框中的"Simulate"按钮进行分析(如按"Esc"键将停止仿真运行)。

举例分析:有一简单电阻串联分压电路如图 5.3.5(a)所示,在直流灵敏度分析中要计算输出节点(2)受电路中元件或电源变化的影响程度。直流灵敏度分析结果如图 5.3.5(b)所示。其理论计算为:节点电压

$$V(2) = [R2/(R1 + R2)] \times V1$$

电阻 R2 对节点 V(2)的灵敏度为

$$dV(2)/dR2 = 0.003$$

电阻 R1 对节点 V(2)的灵敏度为

$$dV(2)/dR1 = -0.003$$

由于当电阻 R1 的阻值增加时，使电路节点 V(2)的电压减小，所以电阻 R1 对节点 V(2)的灵敏度的数值为负数。电压 V1 对节点 V(2)的灵敏度为

$$dV(2)/dV1 = 0.5$$

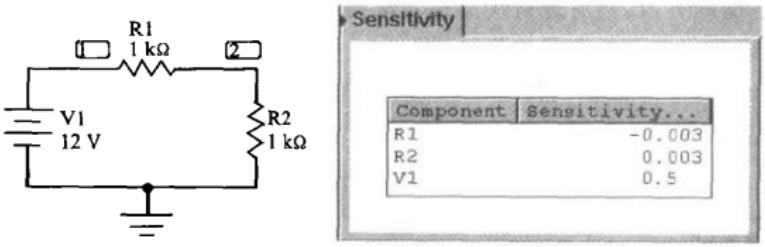


图 5.3.5　分压电路及直流灵敏度分析结果

5.3.6　蒙特卡罗分析

蒙特卡罗(Monte Carlo)分析是一种统计分析，可以用来研究元件参数按选定的误差分布类型在一定范围内变化时，对电路特性的影响。其分析结果可以预测电路在批量生产时的成品率和生产成本。

在进行分析时，首先是在元件参数为常态值情况下来进行仿真，然后其他各次的仿真是在常态值基础上，加减一个随机 δ 值进行运行。该 δ 值由误差分布类型而定。EWB 提供了下列两种分布函数，即

① 均匀分布(平均分布)函数：它是一线性分布函数，所产生的 δ 值是在误差允许度(容差)范围内均匀地产生，任何值是以相等的概率出现。

② 正态高斯分布函数。

蒙特卡罗分析过程如下：

① 选择 Analysis\Monte Carlo 菜单命令，弹出一个对话框。

② 根据对话框要求，设置参数。其参数含义和设置要求见表 5.3.1。

③ 单击对话框中的“Simulate”按钮进行分析(如按“Esc”键将停止仿真运行)。

表 5.3.1　蒙特卡罗分析方法参数设置对话框

蒙特卡罗分析	含义和设置要求
运行次数	必须大于或等于 2
容差(Tolerance)	指均匀分布和正态高斯分布最大允许误差的变化范围。缺省设置：5 %
种子(Seed)	用来启动随机数发生器。缺省设置：0
分布类型	均匀分布/正态高斯分布。缺省设置：均匀分布
输出节点	被分析的节点
扫描……	DC 工作点/瞬态分析/AC 频率分析

5.3.7 最坏状态分析

最坏状态(Worst Case)分析是一种统计分析方法。它可以观察到在元件参数变化时,电路特性变化的最坏可能性。适合于对模拟电路、直流和小信号电路的分析。在分析时,首先进行元件参数常态值的分析,然后进行交流或直流灵敏度分析,在计算出每个参数对输出波形的灵敏度后,就可以获得最坏状态分析的结果。

最坏状态分析产生的数据由排序函数进行收集。该排序函数相当于一个带选择的滤波器,每一次运行只收集一个数据。六个排序函数如表 5.3.2 所列。

表 5.3.2 排序函数

排序函数	含义和设置要求
最大电压	Y 轴最大值
最小电压	Y 轴最小值
在最大处的频率	在 Y 轴最大值处的 X 值
在最小处的频率	在 Y 轴最小值处的 X 值
上升边沿频率	Y 轴值第一次超过用户设定门限值时的 X 值
下降边沿频率	Y 轴值第一次低于用户设定门限值时的 X 值

步骤如下:

① 选择 Analysis\Worst Case 菜单命令,弹出一个对话框。

② 根据对话框要求,设置参数。其参数含义和设置要求如表 5.3.3 所列。

③ 单击对话框中的"Simulate"按钮进行分析(如按"Esc"键将停止仿真运行)。

表 5.3.3 最坏状态分析

最坏状态分析	含义和设置要求
总的容差(Tolerance)	被分析参数的变化值。缺省设置:15 %
排序函数(Collating Function)	当选择 DC 工作点时,只能选最大或最小电压。当选择 AC 频率分析时,可以通过另一对话框来修改分析的参数。缺省设置:最大电压
输出节点	被分析的电压节点
扫描……	DC 工作点/AC 频率分析

中篇　大规模可编程逻辑器件

第六章　可编程逻辑器件概述

在数字系统中大量使用数字逻辑器件，从逻辑功能的特点上可将数字集成电路分为通用型和专用型两大类。数字逻辑电路课程中所介绍的中、小规模数字集成电路都属于通用型，它们的逻辑功能比较简单，而且固定不变。从理论上讲可以用这些通用型数字集成电路组成任何复杂的数字系统；但是需要大量的芯片及芯片连线，且功耗大，体积大以及可靠性差。专用型数字集成电路(ASIC)是为某种专门用途而设计的集成电路。它不仅减小电路体积、重量和功耗，而且使电路的可靠性大为提高；但是在用量不大的情况下，设计和制造的成本很高，而且设计、制造和修定的周期均较长。

可编程逻辑器件(PLD)是20世纪70年代发展起来的新型逻辑器件。它是作为一种通用型器件来生产的，然而它的逻辑功能又是由用户通过对器件编程来自行设定，可以实现在一片PLD芯片上数字系统的集成，而不必由芯片制造厂商去设计和制作专用集成芯片。它是大规模集成电路技术与计算机辅助设计(CAD)、计算机辅助生产(CAM)和计算机辅助测试(CAT)相结合的产物，是现代数字电子系统向超高集成度、超低功耗、超小型化和专用化方向发展的重要基础。

可编程逻辑器件的出现，给数字系统的设计方法带来革命性的变化。通过定义器件内部的逻辑和输入、输出引出端，将原来在电路的印刷线路板设计(PCB)中完成的大部分工作放在芯片设计中进行，减轻了电路图设计和印刷线路板设计的工作量和难度，并且改变了传统的数字系统设计方法，增强了设计的灵活性，提高了工作效率。

6.1　可编程逻辑器件的分类

可编程逻辑器件从编程技术上分为一次性编程和可多次编程。一次性编程在编程后不能修改，采用熔丝工艺制造，一次性编程器件不适合在数字系统的研制、开发和实验阶段使用；而多次编程器件大多采用场效应管作为开关元件，并采用EPROM，E^2PROM，FLASH和SRAM制造工艺生成编程元件，实现器件的多次编程。

可编程逻辑器件从最初的“与”阵列全部预定制PROM到现在复杂的PLD(CPLD，FPGA)器件，大体可分成四个发展阶段，即

第一阶段：PROM，PLA(Programmable Logic Array)；

第二阶段：PAL(Programmable Array Logic)；

第三阶段：GAL(Generic Array Logic)，EPLD；

第四阶段：CPLD，FPGA。

可编程逻辑器件集成密度可分为低密度可编程逻辑器件和高密度可编程逻辑器件。PROM，PLA，PAL 和 GAL 属于低密度可编程逻辑器件；而 EPLD，CPLD 和 FPGA 属于高密度可编程逻辑器件，如图 6.1.1 所示。

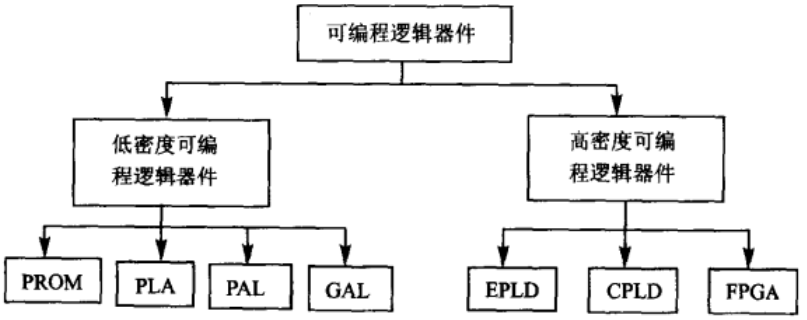


图 6.1.1 可编程逻辑器件的密度分类

1. 可编程只读存储器 PROM

它是 20 世纪 70 年代初期出现的第一代 PLD，其内部结构是由“与阵列”和“或阵列”组成。它可以实现任何“与-或”形式表示的组合逻辑。它采用熔丝工艺编程，只能写一次，不能重复擦写。随着技术发展，出现了 EPROM，E^2PROM 存储器。它们实际上也是一种可编程逻辑器件，具有低价格、易于编程的特点，适合于存储函数和数据表格。

2. 可编程逻辑阵列 PLA

它是 20 世纪 70 年代中期推出的一种基于“与-或阵列”的一次性编程器件，只能用于组合逻辑电路设计，器件内部的资源利用率低，现在已经不常使用。

3. 可编程阵列逻辑 PAL

它是 20 世纪 70 年代末期由 AMD 公司率先推出的一种可编程逻辑器件，由可编程的与逻辑阵列、固定的或逻辑阵列和输出电路三部分组成。它具有多种输出结构形式，可适用于各种组合和时序逻辑电路的设计。但是 PAL 仍采用熔丝编程方式，只能一次性编程而不能重复擦写。

4. 通用阵列逻辑 GAL

它是继 PAL 器件后，在 20 世纪 80 年代初期由 Lattice 公司推出的一种低密度可编程逻辑器件。它在结构上采用了可编程输出逻辑宏单元(OLMC——Output Logic Macro Cell)结构形式。在工艺上吸收了 E^2PROM 的浮栅技术，从而使 GAL 器件具有电可擦写、可重复编程、数据可长期保存和可设置加密位的特点。因此 GAL 器件比 PAL 器件功能更加全面，结构更加灵活，可以取代大部分中、小规模的数字集成电路和 PAL 器件。

5. EPLD 即可擦除的可编程逻辑器件

它是 Altera 公司 20 世纪 80 年代中期推出的一种大规模可编程逻辑器件。它的基本结构形式与 GAL 器件类似，但集成密度比 GAL 器件高得多，EPLD 的输出电路结构采用了与 GAL 相似的可编程输出逻辑宏单元 OLMC，并且大量增加输出逻辑宏单元的数目以及 OLMC 中触发器的预置、置零功能，使 EPLD 具有更大的设计灵活性。

6. CPLD(Complex Programmable Logic Device)即复杂可编程逻辑器件

它是20世纪90年代初由GAL器件发展而来的，采用了COMS EPROM，E^2PROM，FLASH(快闪存储器)和SRAM等编程技术，从而构成了高密度、高速度和低功耗的可编程逻辑器件。其主体仍是与-或阵列，因而称之为阵列型HDPLD。典型的CPLD器件有Lattice公司的PLS/ispLSI系列器件、Xilinx公司的7000和9000系列器件、Altera公司的MAX7000和MAX9000系列器件以及AMD公司的MACH系列器件。

7. FPGA(Field Programmable Gate Array)即现场可编程门阵列

它是1985年由Xilinx公司推出的一种可编程逻辑器件，其电路结构形式与以前的PLD完全不同。它由若干独立的可编程逻辑模块(CLB)排列为阵列组成，通过可编程的内部连线连接这些模块来实现一定的逻辑功能，因而也称之为单元型HDPLD。FPGA的功能由逻辑结构的配置数据决定。这些配置数据存放在片内的SRAM上，所以断电后数据便随之丢失。在工作前需要从芯片外的EPROM中加载配置数据，另外在构成复杂数字系统时一般由若干个CLB组合起来才能实现，而每个信号的传输途径各异，所以信号传输延时时间不能完全确定。

随着微电子技术的发展，可编程逻辑器件的集成度、速度不断提高，目前已达到200万门/片、纳秒级延时水平。其应用越来越广泛，在计算机、通信、智能仪表、医用设备、军事、民用电器等领域得到广泛应用，并成为代表当今电子产品设计变革的主流器件。

6.2　可编程逻辑器件的基本结构

PLD基本结构如图6.2.1所示，由输入缓冲电路、与阵列、或阵列、输出缓冲电路等四部分组成。其中“与阵列”和“或阵列”是PLD器件的主体，逻辑函数靠它们实现。输入缓冲电路主要用来对输入信号进行预处理和提供足够的驱动能力。PLD输出方式有多种，可以由或阵列直接输出(组合方式)，也可以通过寄存器输出(时序方式)；输出可以是低电平或高电平有效，并且有内部通路将输出信号反馈到与阵列输入端。而新型的PLD器件则将输出电路做成宏单元(Macro cell)，使用户可以根据需要选择各种灵活的输出方式(组合方式、时序方式)。众所周知，任何组合逻辑电路均可化为“与-或”式，从而用“与门-或门”二级电路实现；而时序逻辑电路又都是由组合电路加上存储器(触发器)构成的，因此这种PLD结构对实现数字系统具有普遍的意义。

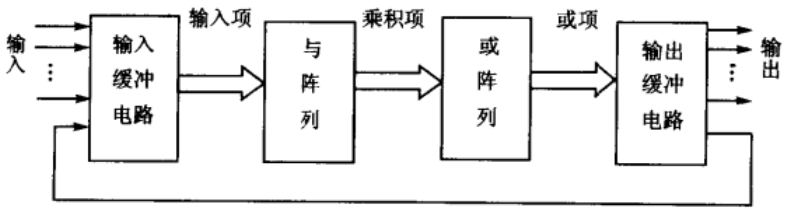


图6.2.1　PLD器件的基本结构框图

6.2.1　PLD电路的逻辑符号表示

可编程逻辑器件内部核心由与阵列和或阵列构成，为了描述PLD的内部电路结构和便于

画图，采用目前国际、国内通行的画法。

图 6.2.2 所示为 PLD 输入缓冲电路表示，提供了原变量和反变量两个互补的输出信号，图中 B=A，C=$\overline{A}$。

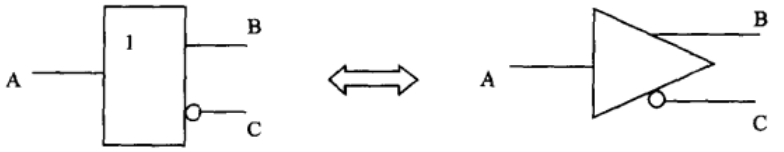
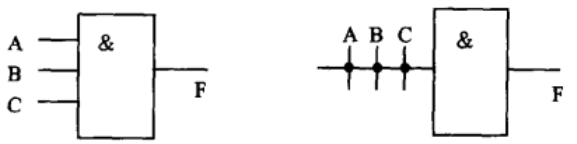


图 6.2.2 PLD 输入缓冲电路

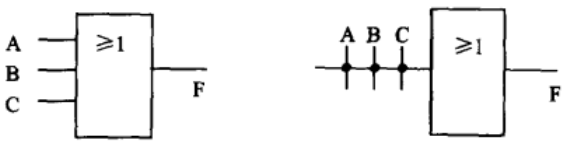
图 6.2.3 所示为一个三输入与门的习惯表示和 PLD 表示，图中 F=A·B·C。



(a) 习惯表示法　　(b) PLD表示法

图 6.2.3 与门表示法

图 6.2.4 所示为一个三输入或门的习惯表示和 PLD 表示，图中 F=A+B+C。



(a) 习惯表示法　　(b) PLD表示法

图 6.2.4 或门表示法

图 6.2.5 所示为 PLD 中阵列交叉点上三种连接方式的表示法，其中固定连接是厂家在生产芯片时已连接，不可编程的；而接通和开断连接是靠编程实现的。在熔丝式工艺的 PLD 中（如 PAL），在尚未编程前交叉点上的熔丝处于接通状态，通过编程可将有用的熔丝保留（连接），将无用的熔丝熔断（断开），即得到所需电路。在 E^2COMS 工艺的 PLD 中每个交叉点上设有 E^2COMS 编程单元，通过编程使该单元处于导通或截止状态，并可多次编程改变状态。

(a) 固定连接　　(b) 可编程连接　　(c) 可编程断开

图 6.2.5 PLD 连接表示法

6.2.2 “与-或”阵列

PLD 器件中最基本的结构是“与-或”阵列，通过编程改变“与阵列”和“或阵列”的内部连接，就可以实现不同的逻辑功能。根据可编程情况可将 PLD 器件分为 PROM，PLA，PAL，

GAL 等四种基本类型。它们内部可编程，如表 6.2.1 所列。

表 6.2.1　PLD 器件的内部可编程

类　型	与阵列	或阵列	输出电路
PROM	固定	可编程	固定
PLA	可编程	可编程	固定
PAL	可编程	固定	固定
GAL	可编程	固定	可组态

PROM 中包括一个固定连接的“与阵列”和一个可编程连接的“或阵列”，其门阵列结构示意图如图 6.2.6 所示。图中 PROM 有 3 个输入和输出，固定“与阵列”中包含了三变量的所有乘积项(2^3个乘积项)。PAL 中包含一个可编程连接的“与阵列”和一个固定的“或阵列”，如图 6.2.7 所示。

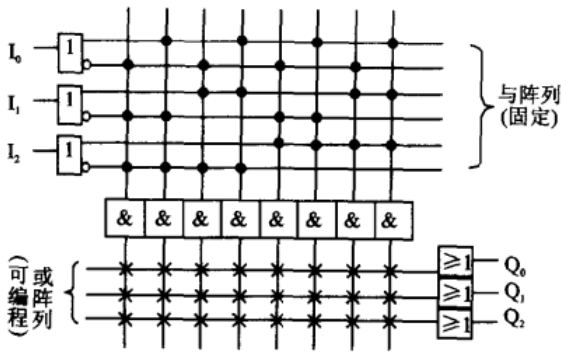


图 6.2.6　PROM 的阵列结构

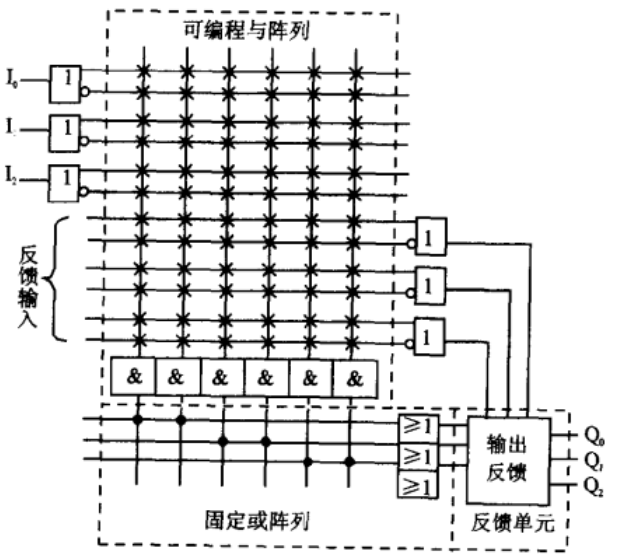


图 6.2.7　PAL 器件的阵列结构

GAL器件和PAL器件的阵列结构是相同的，即“与阵到”是可编程的，“或阵列”是固定连接的。它们之间的差异表现在输出结构上，GAL器件在输出电路中采用了可编程的输出逻辑宏单元(OLMC)，GAL器件结构框图如图6.2.8所示。PAL只能一次编程，而GAL可以实现再次编程，这一点使GAL器件受到用户的欢迎。

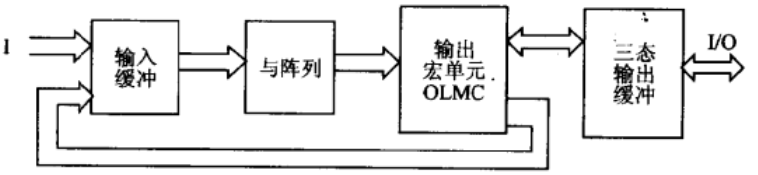


图6.2.8 GAL器件结构框图

6.2.3 逻辑宏单元

PLD器件中的“与-或”阵列只能实现组合逻辑电路的功能，要实现时序逻辑功能则需要有包含触发器或寄存器的逻辑宏单元(OLMC)来实现。逻辑宏单元是高密度可编程逻辑器件中的一个非常重要的基本结构。

Lattice，Altera，Xilinx和AMD等公司在各自生产的PLD产品的宏单元设计上有着各自的特点。总的来看，逻辑宏单元结构具有以下几个方面的特点，即

① 提供时序逻辑需要的触发器或寄存器，并且可以进行各种组态。

② 提供多种形式的I/O方式。

③ 提供内部反馈信号，控制输出的逻辑极性。

④ 分配控制信号，如寄存器的时钟和复位信号，三态门的输出使能信号。

以具有代表性的GAL16V8器件的输出逻辑宏单元(OLMC)为例，介绍逻辑宏单元结构。GAL16V8的OLMC结构如图6.2.9所示。它有1个D触发器、1个8输入“或”门、1个可编程“异或”门和4个可编程多路开关(PTMUX，TSMUX，OMUX，FMUX)。

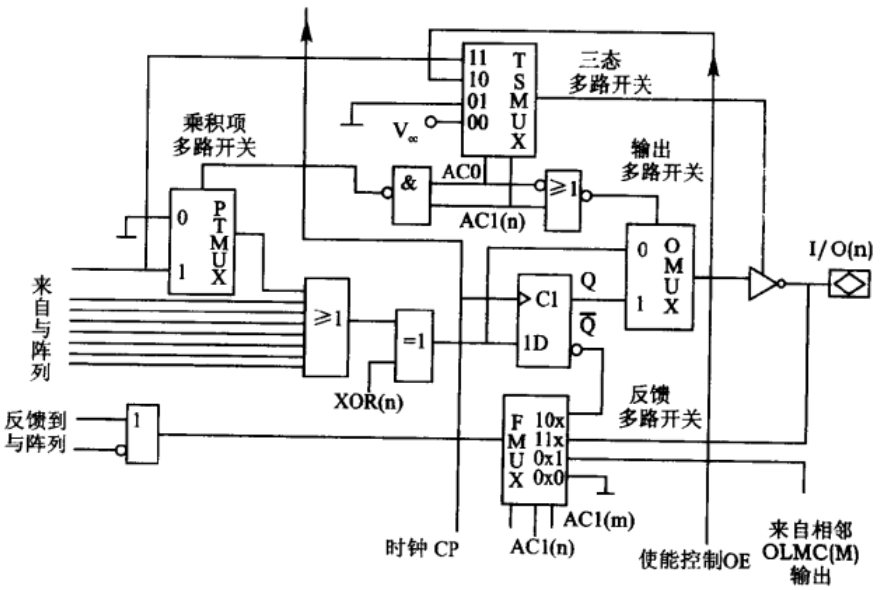


图6.2.9 GAL器件输出逻辑宏单元

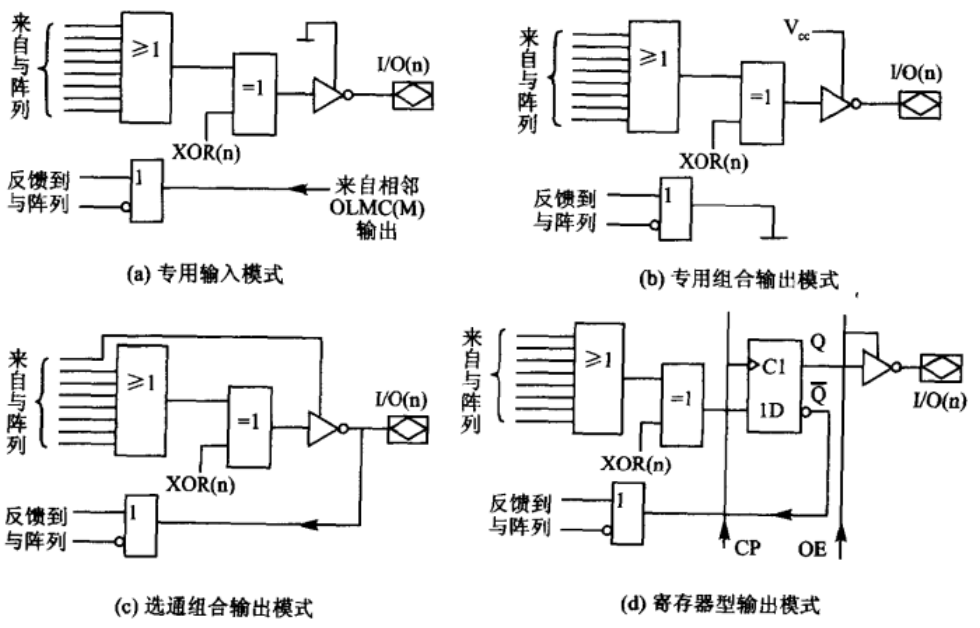
图中的 D 触发器对或门的输出状态起记忆作用；异或门用于控制输出信号的极性。当 XOR(n)＝1 时，异或门起反相器的作用，再经过输出缓冲门的反相，使输出为高电平有效；当 XOR(n)＝0 时，使输出为低电平有效。

输出多路开关(OMUX)是 2 选 1 数据选择器，用于选择输出信号是组合型还是寄存型输出；乘积项多路开关(PTMUX)也是 2 选 1 数据选择器，用于控制来自与阵列的八个乘积输入中的第一个(图中由上而下计算)乘积输入的作用；三态多路开关(OMUX)是 4 选 1 数据选择器，用来控制输出端三态缓冲器的高阻或工作状态；反馈多路开关(FMUX)是 8 选 1 数据选择器，但输入信号只有 4 个，用于决定反馈信号的来源。这些多路开关的状态，取决于结构控制字中 AC0 和 AC1(n)位的值。这可以通过编程来决定。

为了得到不同类型的输出电路结构，只要给 GAL 器件写入不同的结构控制字即可。在 AC0，AC1(n)可编程结构控制位的控制下，输出逻辑宏单元(OLMC)可配置成 4 种不同工作模式，即专用输入(禁止 OLMC 输出)、专用组合输出、选通组合输出、寄存器型输出模式。在表 6.2.2 中列出有关控制位与输出结构的配置关系，图 6.2.10(a)，(b)，(c)，(d)为不同配置模式下 OLMC 的等效电路。

表 6.2.2　OLMC 的工作模式

AC0	AC1(n)	工作模式	等效电路图
0	1	专用输入	如图 6.2.10(a)所示
0	0	专用组合输出	如图 6.2.10(b)所示
1	1	选通组合输出	如图 6.2.10(c)所示
1	0	寄存器型输出	如图 6.2.10(d)所示



(a) 专用输入模式　(b) 专用组合输出模式

(c) 选通组合输出模式　(d) 寄存器型输出模式

图 6.2.10　GAL 器件 OLMC 的 4 种工作模式

6.3 可编程逻辑器件的编程元件

可编程逻辑器件是通过用户编程实现各种逻辑功能，而 PLD 器件中主要有四部分资源可进行编程，即位于芯片中央的可编程功能单元；位于芯片四周的可编程 I/O；分布在芯片各处的可编程布线资源和片内存储块 RAM。各生产厂家对这些可编程元件采用了不同的编程技术，虽然 CPLD 器件和 FPGA 器件均采用 CMOS 技术，但它们在编程工艺上有很大不同，主要有以下四种类型，即

① 熔丝型开关；

② 反熔丝型开关；

③ 浮栅编程元件(EPROM 和 E^2PROM)；

④ 基于 SRAM 的编程元件。

其中前三类是非易失性元件，编程后能使配置数据或开关状态保持在器件上。SRAM 类为易失性元件，即每次掉电后配置数据会丢失。熔丝型开关和反熔丝型开关只能写一次，浮栅编程元件和 SRAM 的编程元件可以进行多次编程。反熔丝型开关一般用在要求较高的军品系列器件上，而浮栅编程元件一般用在民品系列器件上。

6.3.1 熔丝型开关

熔丝型开关是最早的可编程元件，以电流可熔断的熔丝组成。如 PROM，EPLD 和 FPGA 等器件，一般在需要编程的互联节点上设置相应的熔丝开关。在编程时需要保持连接的节点则保留熔丝，需要去除连接的节点则烧断熔丝(断开)，由最后留在器件内的不烧断的熔丝模式决定器件的逻辑功能。

熔丝型开关烧断后不能恢复，只能编程一次，而且熔丝开关很难测试可靠性。在器件编程时，即使发生数量非常小的错误，也会造成器件功能的不正确。为了保证熔丝熔化时产生的金属物质不影响器件的其他部分，熔丝还需要留出极大的保护空间，因此熔丝占用的芯片面积大。

6.3.2 反熔丝型开关

为了克服熔丝型开关的缺点，出现了反熔丝型开关。反熔丝型开关主要通过击穿介质来达到连通线路的目的。在未编程时开关处于开路状态；在编程时其两端加上编程电压，反熔丝就会由高阻抗变为低阻抗，从而实现两个极间的连通，并且编程电压撤除后开关也一直处于导通状态。

Actel 公司采用了一种双极型多层反熔丝工艺的编程元件，称之为 PLICE(可编程低阻抗元素)反熔丝开关，其结构如图 6.3.1 所示。这是一种二端垂直型结构，上面是一层多晶硅，下面是 N^+ 掺杂扩散区，在两者之间是一介质绝缘层。PLICE 反熔丝就生长在这个介质绝缘层上，其生产工艺和 CMOS、双极型、BiMOS 等工艺兼容。它是一种非易失性元件，在未编程时 PLICE 呈现

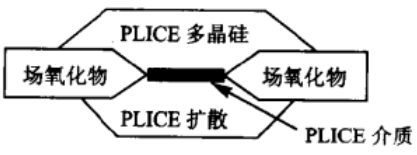


图 6.3.1 PLICE 反熔丝结构

很高的阻抗(>100 MΩ)，当加上 18 V 的编程电压将其击穿后，将建立一个双向的低电阻(约为 100～600 Ω)，反熔丝在硅片上所占面积小于 9 μm^2，电容介于 3～15 pF 之间。因此反熔丝元件占用硅片的面积小，十分适宜于做集成度很高的可编程逻辑器件的编程元件。

除了 Actel 公司的 PLICE 反熔丝工艺外，还有 Quicklogic 生产的非晶体反熔丝技术 Vialink 元件以及 Xilinx 公司推出的采用了最新 Micro Via 工艺的反熔丝结构。

6.3.3　浮栅编程元件

浮栅编程技术包括紫外线擦除电可编程的 EPROM、电擦除电可编程的 E^2PROM 及快闪(Flash 闪速)的存储器，这三种存储器都是用浮栅存储电荷的方法来保存编程的数据，因此在断电后存储的数据不会丢失。

1. EPROM(UVEPROM)

EPROM 的存储内容可以根据需要通过编程更新存储内容，用紫外线擦除原存储内容，由编程重新写入内容。早期 EPROM 结构中的存储单元使用了浮栅雪崩注入 MOS 管，但存在单元面积大、PMOS 管的开关速度低、而且雪崩击穿所需电压较高等缺点。目前多改用叠栅注入 MOS 管(SIMOS 管)制作 EPROM 的存储单元。

图 6.3.2 是 SIMOS 管的结构原理图和符号。它是一个 N 沟道增强型的 MOS 管，有两个重叠的栅极——控制栅 G_c和浮栅 G_f。控制栅 G_c用于控制读出和写入，浮栅 G_f用于长期保存注入电荷。若在漏极和源极之间加上约几十伏的电压脉冲，则在沟道中发生雪崩击穿，令电子加速穿越 SiO_2层注入到浮栅中，从而使浮栅 G_f带上负电荷，该单元相当于存储了“0”。由于浮栅周围都是绝缘的 SiO_2层，泄漏电流极小，所以一旦电子注入到浮栅后，逻辑“0”就能长期保存。当浮栅 G_f无电子积累时，该管相当于存储了“1”。

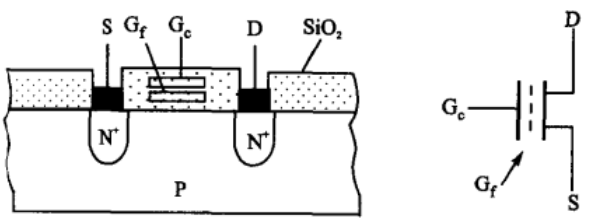


图 6.3.2　SIMOS 管的结构和符号

若要擦去所写入的内容，可用 EPROM 擦洗器产生的强紫外线，穿过 EPROM 芯片上方的石英玻璃窗口，对所有浮栅照射几分钟，使浮栅上的电子获得足够的能量，穿过绝缘层回到衬底中，使浮栅上的电子消失，则芯片又恢复到初始状态，即全部单元都为“1”。

2. E^2PROM

E^2PROM 也可写成 EEPROM，只需在高电压脉冲或在工作电压下就可以进行擦除，而不要借助紫外线照射，所以比 EPROM 更灵活方便，而且它还有字擦除(只擦一个或几个字)功能。

在 E^2PROM 的存储单元中采用了一种叫做浮栅隧道氧化层的 MOS 管。它与 SIMOS 管相似，也有两个栅极。有引出线的栅极为控制栅(也称擦写栅)；无引出线的栅极是浮栅。所不同的是在浮栅与漏极区之间有一小块面积极薄的二氧化硅 SiO_2 绝缘层区域，称为隧道区，并且一个存储单元采用了两只 MOS 管。在控制栅加上几十伏正电压脉冲，就可以在浮栅与漏

极区之间的极薄绝缘层内出现隧道，通过隧道效应，使电子注入浮栅，正脉冲过后，浮栅将长期存储这些电子；若使控制栅接地，则浮栅上的电子通过隧道返回衬底，从而擦除了浮栅内的电子电荷。

显然 E^2PROM 是利用隧道效应使浮栅俘获电子的，与 EPROM 利用雪崩效应不同。一般 E^2PROM 芯片允许擦写 1 万次以上，数据可保存 5～10 年。

3. 快闪存储器

快闪存储器(Flash Memory)是采用一种类似于 EPROM 的单管浮栅结构的存储单元，制成了新一代用电信号擦除的可编程 ROM。它既吸收了 EPROM 结构简单、编程可靠的优点，又具有 E^2PROM 用隧道效应擦除的快捷特性，集成度可以做得很高。

快闪存储器的编程和擦除分别采用两种不同的机理。在编程(写入)方法上，与 EPROM 相似，即利用雪崩注入的方法使浮栅充电；在擦除方法上与 E^2PROM 相似，即利用隧道效应使浮栅上的电子通过隧道返回衬底。由于片内所有叠栅 MOS 管的源极连在一起，所以全部存储单元同时被擦除。这是它不同于 E^2PROM 的一个特点。

早期采用浮栅技术的存储元件都需要使用两种电压，即 5 V 逻辑电压和 12～21 V 的编程电压。现在已趋向采用单电源供电，即由器件内部的升压电路提供编程和擦除电压。大多数单电源可编程逻辑器为 5 V 或 3.3 V 的产品。随着生产工艺水平的提高，这些浮栅编程元件的擦写寿命已达 10 万次以上。

6.3.4 基于 SRAM 的编程元件

SRAM 是指静态存储器。其存储单元由两个 CMOS 反相器和一个用来控制读写的 MOS 管传输开关组成，结构如图 6.3.3 所示。大多数 FPGA 用它来存储配置数据，所以又称为配置存储器。由于采用独特的工艺设计，SRAM 具有很强的抗干扰能力和很高的可靠性。FPGA 中可编程单元的全部工作状态由编程数据存储器中的数据设定。

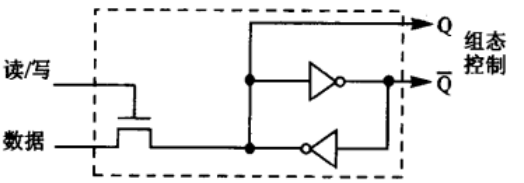


图 6.3.3 SRAM 的基本单元结构

由于 FPGA 中的编程数据存储器是一个静态随机存储器 SRAM 结构，所以断电后存储器中的数据不能保存，因此每次接通电源以后必须重新给存储器装载编程数据，装载过程是在 FPGA 内部的一个时序电路的控制下自动进行的。这些编程数据一般要存放在外加的 EPROM 芯片中，给 FPGA 的使用带来了不便。但基于同样的原因，FPGA 修改器件的逻辑功能很方便，例如在微处理器控制系统中改变 I/O 卡的地址。利用这个特性可以实现功能动态可变的器件和重构式系统。

6.4 边界扫描测试技术

随着微电子技术的发展，器件变得越来越复杂，对器件作全面检测要求也越来越高，而且

越来越重要。一个电路芯片或印制电路板制造出来之后，是否合格，要进行测试。通过测试，判断电路芯片或印制电路板是否存在故障以及故障的位置，以便修复。ASIC 器件生产批量小，功能千变万化，很难用一种固定的测试策略和测试方法来测试其功能。此外，表面安装技术(SMT)和印制电路板制造技术的进步，使得电路板变得越来越小，密度得到提高。这样一来采用传统的测试方法都难于实现，结果是电路板简化所节约的成本，很可能被传统测试方法成本的提高而抵消掉。

为了解决 ASIC 及可编程逻辑器件等超大规模集成电路的测试问题，从 1986 年开始集成电路领域的专家学者成立了“联合测试行动组”，简称为 JTAG(Joint Test Action Group)，已制定了 IEEE1149.1—1990 边界扫描测试技术规范，1990 年被美国电气与电子工程师学会(IEEE)正式承以了。目前大多数高密度的可编程逻辑器件都普遍应用了 JTAG 技术，预计今后新开发的可编程逻辑器件芯片都会支持边界扫描技术。

边界扫描测试 BST(Boundary Scan Test)技术是一种融可测性设计与硅芯片设计为一体的技术。它提供了一种新的完整的方法，克服了测试复杂数字电路板的技术障碍，不需要复杂设备和昂贵的仪器；并且还提供了快速、高效的样品测试及在系统测试，从而减少研制费用，提高了产品质量，促进了新技术(PCB 小型化、复杂 ASIC 和其他 VLSI)的应用。

边界扫描测试结构提供了有效地测试高密度引线器件和高密度电路板上元件的能力。根据 JTAG 制定的边界扫描测试技术规范，标准的边界扫描测试结构如图 6.4.1 所示，主要由以下四个部分组成，即

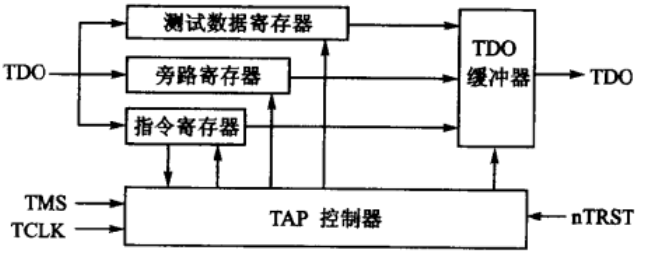


图 6.4.1　边界扫描结构方框图

① 测试数据寄存器：是大型的串行移位寄存器。它位于器件的周边，测试数据是沿着周边作串行移位的，使用 TDI 引脚作为输入，TDO 引脚作为输出。用户可以使用边界扫描寄存器测试外部引脚的连接，或是在器件运行时捕获内部数据。

② 旁路寄存器：是一位数据寄存器，在不进行边界扫描测试时用来旁路通路，使 TDI 和 TDO 端口作为 I/O 引脚。

③ 指令寄存器：是一个 3 位的指令寄存器，用来引导扫描测试数据流，产生测试数据寄存器的控制逻辑。边界扫描测试指令分为公开(必选)指令、可选用指令和各公司私有指令三类，其中公开指令有三条：抽样/预加载、外测试和旁路。指令代码是由 TDI 引脚在时钟控制下送入，以确定指令模式。

④ 测试访问端口 TAP(Test Access Port)控制器：是一个 16 态的状态机，在 TCLK 的上升沿时刻进行状态转换，控制管理 JTAG 的操作。

边界扫描测试 JTAG 电路主要有 5 个引脚。其功能说明如表 6.4.1 所列。

表 6.4.1 边界扫描测试 JTAG 电路的引脚名称及功能说明

引 脚	名 称	功能说明
TDI	测试数据输入	指令和测试数据的串行输入引脚。在 TCLK 的上升沿时刻移入
TDO	测试数据输出	指令和测试数据的串行输出引脚。在 TCLK 的下降沿时刻移出。如果没有数据移出,则此引脚处于三态(高阻)
TMS	测试模式选择	选择 JTAG 指令模式的串行输入引脚。在 TCLK 的上升沿时刻移入。在用户状态下 TMS 应置于高电平
TCLK	测试时钟输入	时钟引脚。用来将串行数据和指令分别移入 TDI 和移出 TDO 引脚。也用于把串行指令和数据移入 TMS 引脚
nTRST	测试复位输入	低电平有效,用于异步初始化或复位边界扫描电路

当器件工作在 JTAG BST 模式时,复位引脚 nTRST 应置于高电平,然后使用其他 4 个引脚进行边界扫描测试操作。复位引脚 nTRST 为低电平,能够异步地把边界扫描测试电路初始化或复位。当器件不工作在 JTAG BST 模式时,nTRST 应置于低电平(以保持 JTAG 电路已完成初始化),TDI,TDO 和 TCLK 应当维持在低电平,而 TMS 应当置于高电平。

第七章　Altera 和 Lattice 公司的可编程逻辑器件

7.1　Altera 公司的可编程逻辑器件简介

美国的 Altera 公司是一家比较新的可编程逻辑器件生产厂家，以其雄厚的技术实力，独特的设计构思和功能齐全的芯片系列，成为生产 PLD 的主要厂家之一。因此本书选择 Altera 公司的可编程逻辑器件作为典型器件，以说明可编程逻辑器件的构造、性能和设计方法。

7.1.1　Altera 公司的产品发展过程

首先来看一下 Altera 公司的产品发展过程。

1983—1985 年，Altera 公司宣告成立，并且生产出世界上第一片可擦除、可编程逻辑器件 EP300 和高密度可编程器件 EP1200。

1988 年，Altera 公司开始生产采用多阵列(MAX)结构的 MAX5000 系列芯片。

1989—1991 年，Altera 公司推出世界上第一家基于 Windows 的开发工具软件 MAX＋PLUSⅡ和自己开发的硬件描述语言 AHDL，同时，MAX7000 系列芯片开始上市。

1992 年，Altera 公司生产出它的第一片采用 E^2PROM 工艺的 PLD 产品 EPM7032，同时，推出当时最高密度可编程器件 FLEX8000 系列。

1993 年，推出世界上第一片采用 3.3 V 电源的 CPLD 产品 EPM7032V，同时，Altera 公司的开发系统软件被售出 15 000 套。

1994 年，Altera 公司购买 Intel 公司的可编程器件事业部，MAX9000 系列开始上市。

1995 年，MAX7000S 新结构面市，并且开始推出 FLEX10K 芯片系列。

1996 年，世界上第一片 10 万门级的 PLD 芯片 EPF10K100 在 Altera 公司面市，FIEX10KA 系列产品开始上市。

1997 年，Altera 公司推出 FLEX6000 结构和 MAX7000A 结构的产品，同时宣布开发 200 万门级的 Raphael 芯片系列。

1998 年，Altera 公司宣布开发出 3.3 V 电源的在系统可编程(ISP)逻辑器件。同时推出业界最大规模的嵌入式 PLD 产品 EPF10K250A 系列。

1999 年，Altera 公司推出采用多核(Multi Core)结构的 APEX20K 系列，其等效门数达到 10 万～100 万门级。

7.1.2　Altera 公司的可编程逻辑器件系列

Altera 公司的产品可分为如下系列：Classic 系列、MAX(Multiple Array Matrix)系列、FLEX(Flexible Logic Element Matrix)系列、APEX 系列以及最近推出的 ACE 系列。通过该公司先进的开发工具软件 MAX＋PLUSⅡ(Multiple Array Matrix＋Programmable Logic User SystemⅡ)，用户可以对芯片进行编程、加密或用软件代替硬件，以满足设计要求。特别

是在系统可编程逻辑器件的出现，为可编程器件的数据擦除和重新配置提供了方便，从而大大提高了设计速度。

Altera公司的产品基本上属于CPLD结构。它的内部连线均采用集总式互联通路结构，即利用同样长度的一些连线实现逻辑单元之间的互联。这种结构的互联是集总式，任意两逻辑单元之间的延时是相等并可预测的。Altera公司的FIEX系列芯片同时具有FPGA和EPLD两种结构的优点，得到较广泛的应用。图7.1.1为Altera器件内部互联结构的演变过程。Classic类型的集成度较低，采用全局连线，基于EPROM工艺并可用紫外线擦除和多次编程；MAX5000系列采用了可编程连线阵列和EPROM编程工艺；MAX7000系列采用增强型可编程连线阵列，可用门为600～5 000个，具有32～256个宏单元和36～164个用户I/O引脚，采用E^2PROM工艺；而后来推出的FLEX8000，MAX9000，FLEX10K则大都采用快速通道(Fast Track)连接结构，采用SRAM工艺，使其维持状态的功耗很低，并可进行在线重新配置。表7.1.1所列为Altera器件的内部结构和工艺。

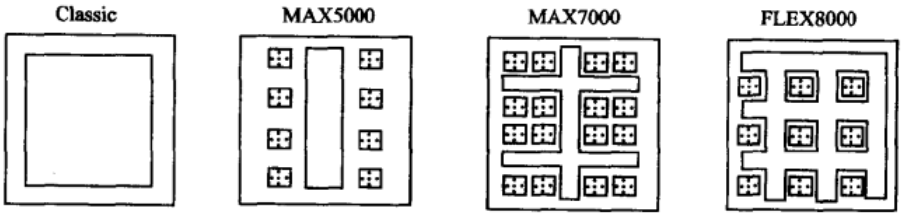


图7.1.1 Altera器件内部结构的演变

表7.1.1 Altera器件的结构和工艺

器件系列	逻辑单元	连线结构	工 艺
Classic	积之和	连续式	EPROM
MAX5000	积之和	连续式	EPROM
MAX7000	积之和	连续式	E^2PROM
MAX9000	积之和	连续式	E^2PROM
FLEX6000	查找表	连续式	SRAM
FLEX8000	查找表	连续式	SRAM
FLEX10K	查找表	连续式	SRAM
APEX20K	查找表	连续式	SRAM
ACE	查找表	连续式	SRAM

MAX和FLEX系列芯片是Altera公司目前最为流行、使用最广泛的两类器件。本章着重介绍这两类系列芯片的电路结构、编程和设计方法。APEX系列是比较新的器件。它是采用多核结构，着眼于系统级的设计而推出的一种芯片，能够真正地实现可编程片上系统SOPC(System On a Programmable Chip)，采用0.18 μm COMS SRAM工艺规程，集成度最高可达到200万门。ACE则是1999年Altera公司着眼于通信(如调制解调、路由器、运程接收系

统)、音频处理及其他应用而推出的新的芯片系列。其速度最高可以达到 160 MHz,功耗极低,器件的密度为 20 000～150 000 门;在结构上具有嵌入式存储态、支持 64 bit、66 MHz 的 PCI 总线、支持锁相环等特点。

7.2　Altera 公司的可编程逻辑器件结构特点

本节着重介绍 MAX7000 和 FLEX10K 这两类系列芯片的电路结构和特点。

7.2.1　MAX7000 系列器件

1. MAX7000 系列器件特点

MAX7000 系列是高性能、高密度的 CMOS CPLD,在制造工艺上采用 0.8 μm CMOS E^2PROM 技术。其中 MAX7000 系列包含了多种不同类型的器件,其主要性能指标如表 7.2.1 所列。MAX7000 系列器件的主要特点如下:

表 7.2.1　MAX7000 器件典型数据

特　性	EPM7032	EPM7064	EPM7096	EPM7128	EPM7160	EPM7192	EPM7256
集成门数	1 200	2 500	3 600	5 000	6 400	7 500	10 000
可用门数	600	1 250	1 800	2 500	3 200	3 750	5 000
宏单元数	32	64	96	128	160	192	256
逻辑阵列块	2	4	6	8	10	12	16
I/O 引脚数	36	36,38,40,68	52,64,76	68,84,100	68,84,104	124	84,120,164
t_{PP}/ns	5	5	6	6	6	7.5	7.5
t_{SU}/ns	4	4	5	5	5	6	6
t_{FSU}/ns	2.5	2.5	2.5	2.5	2.5	3	3
t_{CO}/ns	3.5	3.5	4	4	4	4.5	4.5
f_{CNT}/MHz	151.5	151.5	151.5	125	125	100	100

① 高性能可擦除器件,采用第二代多阵列矩阵(MAX)结构。
② 集成密度门数可达 10 000 门,可用门数为 600～5 000 门。
③ 引脚之间的延时为 6 ns,可达最高 151.5 MHz 的工作频率。
④ MAX7000S 系列通过标准的 JTAG 接口,支持在系统编程(ISP)。
⑤ 高性能的可编程连线阵列(PIA)提供一个高速的、延时可预测的五连资源网络。
⑥ 每个宏单元(MC)中可编程扩展乘积项(P－Terms)可达 32 个。
⑦ 具有全面保护设计的可编程保密位。
⑧ 具有独立的全局时钟信号。
⑨ 可提供 2.5 V(MAX7000B),3.3 V(MAX7000A),5.0 V(MAX7000S)电源供电。

在 MAX7000 系列中，MAX7000E 类型的器件是高密度的，其中包括：EPM7128E，EPM7160E，EPM7192E 和 EPM7256E。这些器件有几项功能得到加强，例如附加全局时钟、附加输出使能控制以及增加连线资源、快速输入寄存器等等。

MAX7000S 和 MAX7000A 类型的器件具有在系统可编程(ISP)功能，包括：EPM7032S，7064S，7128S，7160S，7192S，7256S 和 EPM7128A，7256A 等器件。除了 ISP 功能外，还包含 JTAG 边界扫描测试(BST)电路和更多的宏单元等。

MAX7000 芯片在结构上包含有 32～256 个宏单元，每 16 个宏单元组成一个逻辑阵列块(LAB)，共有 2～16 个 LAB。每个宏单元有一个可编程的“与”阵列和一个固定的“或”阵列，以及一个触发器，这个触发器具有独立可编程的时钟、时钟使能、清除和置位等功能。为了能构成复杂的逻辑函数，每个宏单元可使用共享扩展乘积项和高速并联扩展乘积项，总的向每个宏单元提供了 32 个乘积项。

2. MAX7000 系列器件的结构

图 7.2.1 示出 MAX7000 系列器件的结构方框图，其结构主要由以下部分组成，即

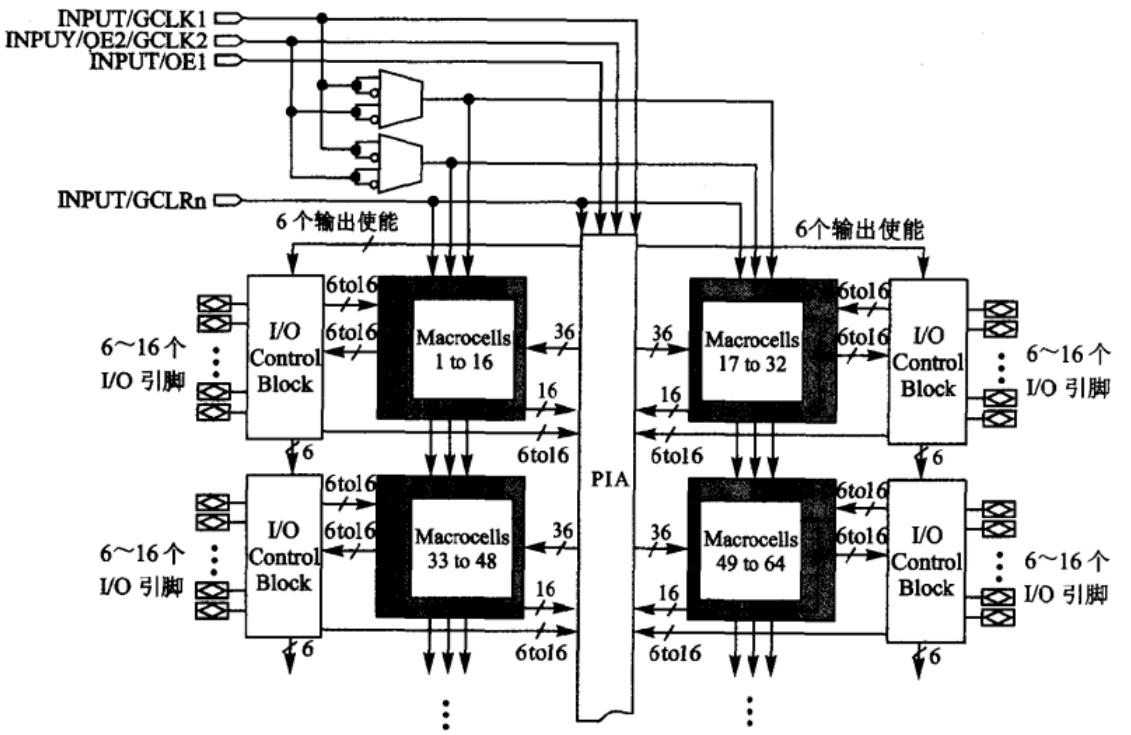


图 7.2.1 MAX7000 系列器件的结构框图

① 逻辑阵列块 LAB(Logic Array Blocks)；

② 宏单元(Macrocells)；

③ 扩展乘积项(共享和并联)(Expender Product Terms)；

④ 可编程连线阵列 PIA(Programmable Interconnect Array)；

⑤ I/O 控制块(I/O Control Blocks)。

每个器件包含了四个专用输入，可用作通用输入，也可作为每个宏单元和 I/O 引脚的高

速、全局控制信号，如时钟(Clock)、清除(Clear)和输出使能(OE)。

(1) 逻辑阵列块 LAB

MAX7000 器件的结构主要由逻辑阵列块 LAB 和它们之间的连线构成。每个逻辑阵列块由 16 个宏单元组成，多个 LAB 通过可编程连线阵列 PIA 和全局总线连接在一起。全局总线由所有的专用输入、I/O 引脚和宏单元馈给信号。LAB 的输入信号有：① 来自 PIA 的 36 个信号；② 全局控制信号；③ I/O 引脚到寄存器的直接输入通道。

(2) 宏单元

MAX7000 宏单元由逻辑阵列、乘积项选择矩阵和可编程触发器三个功能块组成。宏单元的结构框图如图 7.2.2 所示。

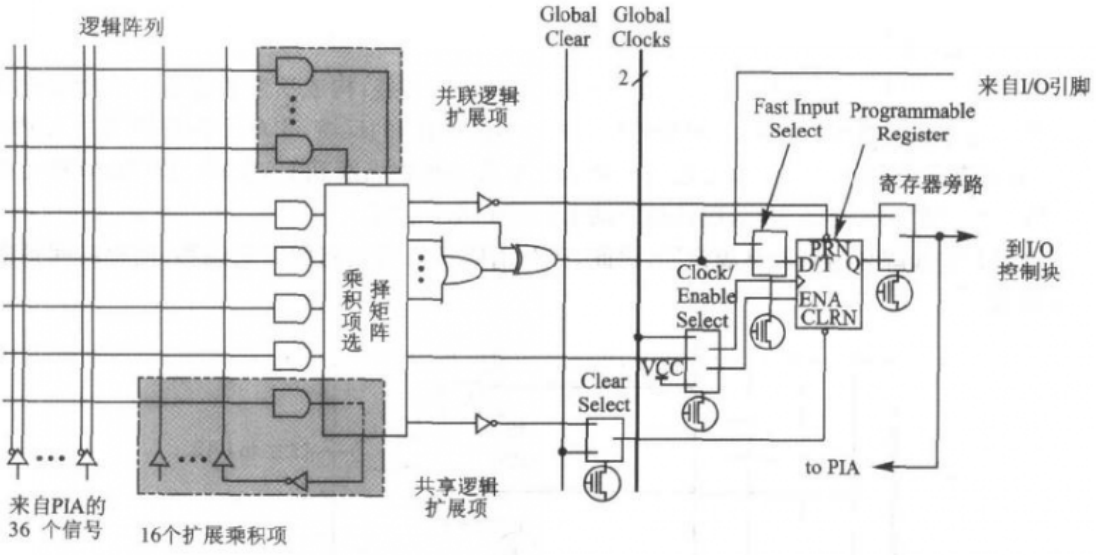


图 7.2.2　宏单元结构框图

逻辑阵列实现组合逻辑功能，给每个宏单元提供 5 个乘积项。“乘积项选择矩阵”分配这些乘积项作为到“或”门和“异或”门的主要逻辑输入，以实现组合逻辑函数，或者把这些乘积项作为宏单元中触发器的辅助输入，即清除、置位、时钟和时钟使能控制。每个宏单元的一个乘积项可以反相后回送到逻辑阵列。这个“可共享”的乘积项能够连接到同一个 LAB 中任何其他乘积项上。利用 MAX+PLUSⅡ开发工具按设计要求自动优化乘积项的分配。

宏单元中的触发器可以单独地编程为具有可编程时钟控制的 D 触发器、T 触发器、SR 触发器或 JK 触发器工作方式。另外，也可以将触发器旁路，实现组合逻辑功能。每个触发器也支持异步清除和异步置位功能，乘积项选择矩阵分配乘积项去控制这些操作。虽然乘积项驱动触发器的置位和复位信号是高电平有效，但是在逻辑阵列中可将信号反相，得到低电平有效的控制。此外，每个触发器的复位功能可以由低电平有效的、专用的全局复位引脚 GCLRn 信号提供。在设计输入时，用户可以选择所希望的触发器，然后 MAX+PLUSⅡ开发工具对每一个寄存器功能选择最有效的触发器工作方式，以使设计所需要的资源最少。

(3) 扩展乘积项

大多数逻辑函数虽然能够用宏单元中的 5 个乘积项来实现，但某些逻辑函数较为复杂，需

要附加乘积项。为提供所需的逻辑资源，不利用另一个宏单元，而是利用 MAX7000 结构中共享和并联扩展乘积项，作为附加的乘积项直接送到本 LAB 的任意宏单元中。在实现逻辑综合时，利用扩展项可保证用尽可能少的逻辑资源，实现尽可能快的工作速度。

共享扩展乘积项在每个 LAB 中有 16 个扩展项。它是由宏单元提供的一个未使用的乘积项，并把它们反馈到逻辑阵列，便于集中管理使用。每个共享扩展乘积项可被 LAB 内任何(或全部)宏单元使用和共享，以实现复杂的逻辑函数。

并联扩展项是一些宏单元中没有使用的乘积项，并且这些乘积项可分配到邻近的宏单元去实现快速复杂的逻辑函数。并联扩展项允许多达 20 个乘积项直接馈送到宏单元的“或”逻辑，其中 5 个乘积项是由宏单元本身提供的，15 个并联扩展项是由 LAB 中邻近宏单元提供的。

(4) 可编程连线阵列(PIA)

通过这个 PIA 的可编程布线通道，把多个 LAB 相互连接，构成所需的逻辑。它能够把器件中任何信号源连接到目的地。所有的专用输入、I/O 引脚的反馈、宏单元的反馈均连入 PIA 中，并且布满整个器件。图 7.2.3 示出了 PIA 的信号是如何布线到 LAB 的。EPROM 单元控制 2 输入“与门”的一个输入端，以选择驱动 LAB 的 PIA 信号。

可编程连线阵列的延时是固定的，因此，PIA 消除了信号之间的时间偏移，使得时间性能容易预测。

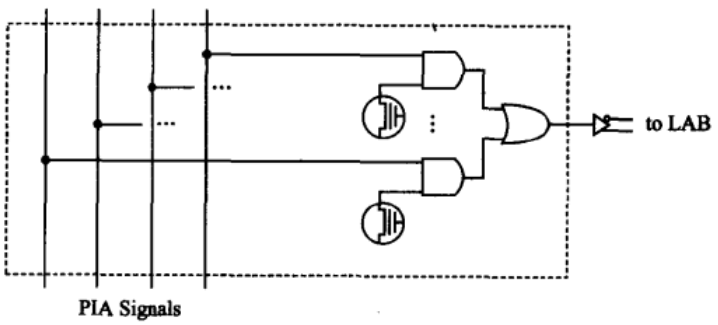


图 7.2.3 PIA 布线图

(5) I/O 控制块

I/O 控制块允许每个 I/O 引脚单独地配置成输入、输出和双向工作方式，所有 I/O 引脚都有一个三态缓冲器。它的使能端由 OE1n，OE2n 及 VCC，GND 信号中的一个控制。I/O 控制块结构图如图 7.2.4 所示。该 I/O 控制块由两个全局输出使能信号 OE1n 和 OE2n 来驱动。当三态缓冲器的控制端接到地(GND)时，其输出为三态(高阻态)，而且 I/O 引脚可作为专用输入端使用；当三态缓冲器的控制端接到电源(VCC)时，I/O 引脚处于输出工作方式。

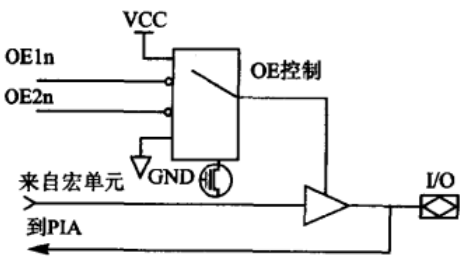


图 7.2.4 I/O 控制块结构图

7.2.2　FLEX10K 系列器件

1. FLEX10K 系列器件特点

FLEX(灵活逻辑单元矩阵)系列是 Altera 公司近几年推出的产品,包括 FLEX8000 系列、FLEX10K 系列和 APEX20K 系列等。它们具有高的集成度和丰富的寄存器资源,采用快速通道的连续式布线结构,是一种将 CPLD 和 FPGA 的优点结合于一体的新型器件,目前已得到广泛的应用。

FLEX10K 系列器件是在 FLEX8000 器件基础上发展起来的,与 MAX7000 系列器件不同的是,它引入了一种逻辑单元(LE)大矩阵,每 8 个组成一组构成一个逻辑阵列块,即 LAB。另外采用了嵌入式阵列块(EAB),组成嵌入式存储器。

FLEX10K 系列器件的主要特点如下:

① 高密度、SRAM 工艺制造,10 000～250 000 典型门;

② 功能更强的 I/O 引脚,每个引脚都是独立的三态门结构,具有可编程的速率控制;

③ 嵌入式阵列块(EAB),每个 EAB 提供 2 Kbit;

④ 逻辑单元(LE)采用查找表(LUT)结构;

⑤ 采用快速通道(Fast Track)互联布线结构;

⑥ 实现快速加法器和计数器的专用进位链;

⑦ 实现高速、多输入逻辑函数的专用级联链。

FLEX10K 系列芯片的主要数据如表 7.2.2 所列。

表 7.2.2　FLEX10K 芯片的数据

特　性	EPF10K10 EPF10K10A	EPF10K30 EPF10K30A EPF10K30E	EPF10K50 EPF10K50V EPF10K50E EPF10K50S	EPF10K100 EPF10K100A EPF10K100B EPF10K100E	EPF10K200E EPF10K200S	EPF10K250A
典型门数	10 000	30 000	50 000	100 000	200 000	250 000
逻辑单元	576	1 728	2 880	4 992	9 984	12 160
逻辑阵列块 LAB	72	216	360	624	1 248	1 520
嵌入式阵列块 EAB	3	6	10	12	24	20
RAM/bits	6 144	12 288 24 576	20 480 40 960	24 576 49 152	98 304	40 960
最大 I/O 引脚	134	246	310	406	470	470

FLEX10K 系列器件也提供多种电源电压,如 2.5 V(FLEX10KE),3.3 V(FLEX10KA),5.0 V(FLEX10KE);以及各种封装形式,如 PLCC(Plastic J - Lead Chip Carrier),TQFP

(Thin Quad Flat Pack),PQFP(Plastic Quad Flat Pack),RQFP(POWER Quad Flat Pack),BGA(Ball-Grid Array),PGA(Pin-Grid Array)。

2. FLEX10K 器件的结构

FLEX10K 结构如图 7.2.5 所示,主要包括:逻辑阵列块(LAB)、快速通道(Fast Track)互联、嵌入式阵列块(EAB)和 I/O 单元(IOE)这四部分。逻辑阵列块由 8 个逻辑单元(LE)和一个局部互联组成,所有的 LAB 按行和列排成一个矩阵,在每一行中放置一个 EAB。器件中信号的连接、进出器件的引脚以及多个 LAB 的互联均采用快速通道互相连接。在每行(每列)快速通道互联线的两端连接着 I/O 单元。

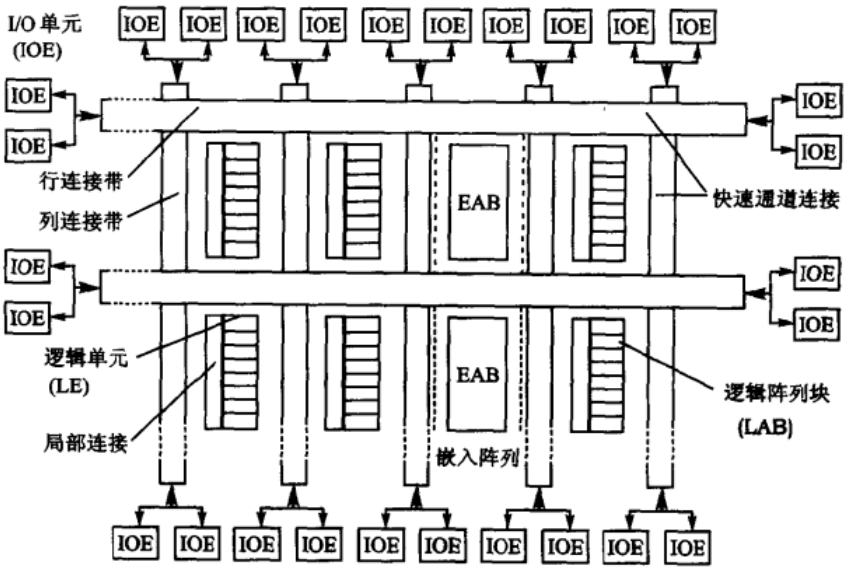


图 7.2.5 FLEX10K 结构框图

(1) 逻辑阵列块(LAB)

逻辑阵列块构成 FLEX10K 芯片结构的主体部分。它由 8 个 LE、与 LE 相连的进位链和级联链、LAB 控制信号以及局部互联线组成。

每个 LAB 包含有 4 个可使用的控制信号,其中有两个可用作时钟,另外两个用作清除和置位逻辑控制。这些控制信号可由专用输入引脚、I/O 引脚或借助 LAB 局部互联的任何内部信号直接驱动,专用输入端一般用作公共的时钟、清除或置位信号。

(2) 逻辑单元(LE)

每个 LE 由四输入查找表(LUT)、一个可编程寄存器和具有进位、级联功能的信号通道所组成,并且有两个输出,以驱动局部互联和快速通道互联,可实现组合逻辑和时序逻辑功能。逻辑单元的结构框图如图 7.2.6 所示。

在逻辑单元中的 LUT 和寄存器能分别被用于不相关的功能。寄存器的数据输入端能被 LUT 的输出驱动,也可由 DATA4 信号直接驱动。LUT 和寄存器的输出可分别由 LE 的两个输出端同时输出。它能够有效地提高 LE 的利用率。

LUT 是一个函数发生器,能快速计算四输入变量的任意函数。LE 中的可编程触发器可设置成 RS 触发器、T 触发器、D 触发器和 JK 触发器。该触发器的时钟、清除和置位控制信号

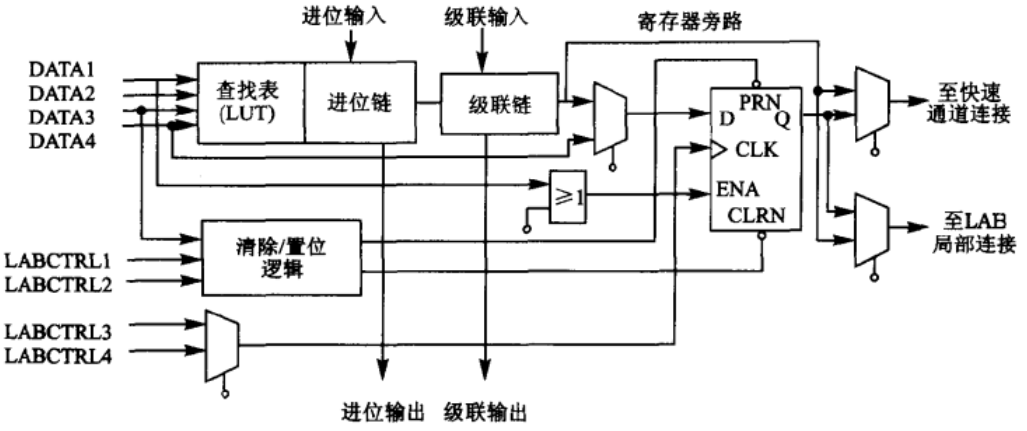


图 7.2.6　FLEX10K 逻辑单元的结构框图

可由专用输入引脚、I/O 引脚或任何内部逻辑驱动。可将 LE 中的寄存器(触发器)旁路、使 LUT 的输出直接连接到 LE 的输出,以实现纯组合逻辑函数。

FLEX10K 在结构上还提供了两条专用快速通路,即进位链和级联链。它们连接相邻的 LE,不占用快速通道互联通路。进位链提供 LE 之间非常快(小于 1 ns)的向上进位功能,来自低位的进位信号由进位链向上送到高位,同时送到 LUT 和进位链的下一段。因此用进位链可实现高速计数器和任意位数的加法器。

利用级联链可以实现输入变量很多的逻辑函数。相邻的 LUT 可并行地计算函数的各个部分,级联链起着把计算的中间结果串联起来的作用。可以使用逻辑"与"或者逻辑"或"来连接相邻 LE 的输出。每增加一个 LE,逻辑函数的有效输入个数增加 4 个,如 n 个 LE 的级联链可实现 4n 个输入变量的逻辑函数,但其延时大约会增加 1 ns。Altera 公司的 MAX+PLUSⅡ开发工具在编译过程中能够自动建立进位链和级联链,设计者也可以用手工插入进位链和级联链。

(3) 嵌入式阵列块(EAB)

嵌入式阵列块是 FLEX10K 系列器件在结构设计上的一个重要部件。它是一种在输入端口和输出端口都带有寄存器的非常灵活的 RAM 块,既可以用作存储器使用,也可以用来实现逻辑功能。

当作为存储器使用时,每个 EAB 可提供 2 048 bit,可用来构成 RAM,ROM,FIFO RAM 或双端口 RAM。每个 EAB 单独使用时,可配置成以下几种规格:256×8,512×4,1 024×2 或 2 048×1 等。通过连接多个 EAB 可组合成为一个规模更大的 RAM,例如,两个 256×8 的 RAB 可组成一个 256×16 的 RAM(位扩展);两个 512×4 的 RAB 可组成一个 512×16 的 RAM(位扩展)。如果需要,在器件中可以把 EAB 级联成一个容量更大的 RAM(字扩展),如可以把 EAB 级联成达到 2 048 字节的 RAM。这些 RAM 的组合形式可通过 MAX+PLUSⅡ开发工具按设计人员的要求自动组合。

嵌入式阵列块也可用于实现复杂的逻辑功能,此时每个 EAB 可相当于 100～300 个等效门,能够方便地构成乘法器、加法器、纠错电路等模块,并由这些功能模块进一步构成诸如数字滤波器和微控制器等功能的系统。逻辑功能通过配置后,可编程 EAB 为只读模式,并且可生

成一个大的查找表(LUT),在这个 LUT 中组合逻辑功能通过查找表而不是通过计算来实现,其执行速度比通常在逻辑里应用算法执行要快。而且输入、EAB 容量使得设计者能够在一个逻辑级上完成复杂的逻辑功能,避免了多个 LE 连接带来的连线延时。

EAB 为驱动和控制信号提供了灵活的选择。对于 EAB 的各种输入和输出可采用不同的时钟,全局信号、专用时钟引脚和局部互联均可以驱动时钟。寄存器能被独立地嵌入在数据输入、EAB 输出、地址和写使能(WE)信号上。全局信号和 EAB 局部连接可以驱动写使能(WE)信号。由于 LE 可驱动 EAB 局部互联,所以 LE 能够控制 WE 或 EAB 时钟信号。

每个 EAB 由行连线馈入信号,其输出可传输到行连线和列连线上。每个 EAB 的输出可以驱动两个行通道中的任何一个和两个列通道中的任何一个。未利用的行通道可被另一个列通道驱动,这一特性为 EAB 的各种输出增加了布线资源的可利用性。

(4) 快速通道互联

FLEX10K 结构的另一个特点是:在器件内部信号的互联是由快速通道(Fast Track)连线提供的。它是贯穿整个器件内的一系列水平和垂直的连续式布线通道。快速通道由"行连线带"和"列连线带"组成,每条"行连线带"和"列连线带"的两端都设有 I/O 单元(IOE),与I/O 引脚连接。采用这种布线结构,即使对于复杂的设计也可以预测其性能,而 FPGA 中的分段式连线结构需要用一些开矩阵把数目不同的若干线段连接起来,这就增加了逻辑资源间的延时,使器件性能下降。

器件内部的 LAB 按行和列组成一个矩阵,每行 LAB 由"行连线带"连接,"行连线带"由上百条"行通道"组成。这些通道水平地贯通整个器件,承载进、出这些行中 LAB 的信号。行连线带可以驱动 I/O 引脚或馈送至其他 LAB。

"列连线带"由几十条"列通道"组成。LAB 中的每个 LE 最多可驱动两条独立的列通道,列通道垂直地贯通整个器件,不同行中的 LAB 借助局部的多路选择开关共享这些资源。

(5) 输入/输出单元(IOE)

每个 IOE 包含一个双向 I/O 缓冲器和一个输入/输出寄存器,可用作输入、输出或双向引脚。每个引脚可被设置为集电极开路输出方式。

IOE 中的时钟、清除、时钟使能和输出使能由被称作周边控制总线的 I/O 控制信号网络提供。如果要求多于 6 个时钟使能或多于 8 个输出使能信号,则 IOE 能由特定 LE 的输出得到时钟使能或输出使能控制。IOE 中的整片输出使能引脚是低电平有效,可用来使器件上所有引脚变成三态。该选项可在设计文件中设置。

以上简单介绍了 FLEX10K 系列器件的结构特点和内部逻辑功能。由于 FLEX10K 器件采用 SRAM 工艺生产,所以其内部逻辑功能和连线设置由芯片内 SRAM 所存储的数据决定。芯片加电时,通过存储在芯片外部的串行 EPROM 或者由系统控制器提供的数据对 FLEX10K 器件进行编程。Altera 公司的 EPC1441 和 EPC1 是专门供 FLEX10K 器件配置数据用的 EPROM,借助串行数据流配置 FLEX10K。配置数据也可以配置在其他的 EPROM 中,或者系统的 RAM 通过 Altera 公司的 BitBlaster 串行下载电缆下载到 FLEX10K 中。配置完成后,还可以通过复位进行在线重新配置,装入新数据,实现新功能。重新配置所需时间很短(小于 100 ms),因此在系统工作过程中可以实时地改变配置。

7.3 Lattice公司的在系统编程器件简介

美国的Lattice(莱迪思)公司是较早利用E^2COMS技术制造可编程逻辑器件的公司之一,是世界上第一片GAL的研制者。近年来,该公司在HDPLD的研制方面取得很大进展,特别是于1991年发明并率先推出在系统编程(ISP)技术,开拓了新一代的PLD。Lattice公司的可编程逻辑器件主要包括:高密度可编程逻辑器件ispLSI&MACH、低密度可编程逻辑器件ispGAL、信号开关/接口器件ispGDX、ispGDS和可编程模拟电路ispPAC等系列产品。

ispPAC系列器件是Lattice公司推出的业界第一个高性能在系统可编程模拟集成电路,高度集成、精确的设计使得用一个ispPAC就可代替几十个标准分立元件,并且提供了相应的PAC-Designer开发工具软件。在系统可编程模拟电路为用户提供了一个强大的新的模拟电路的设计、集成和构建的方法,使设计者第一次可以不依赖于那些不确定费用和开发周期的ASIC器件,而方便地实现他们特有的设计。

高密度可编程逻辑器件ispLSI是Lattice公司的主流产品,本节主要介绍数字式ISP系列器件的结构、特性及其设计编程。表7.3.1列出了ISP系列器件的主要特性。

表7.3.1 ISP系列器件的主要特性

特 性	ispLSI	ispGAL	ispGDX	ispGDS
器件	1000/E,2000,2000E 2000,2000VE,3000 5000V,6000,8000	ispGAL22V10 22LV10	在系统可编程通用数字互联	在系统可编程开关矩阵
密度	1 000～50 000门	500门	80×80～16×15	7×7～11×11
宏单元数	32～850	10	—	—
寄存器数	32～1 152	10	—	—
I/O端口	34～329	22	80～160	14～22
F_{max}/MHz	225	167(3.3 V) 111(5 V)	250(3.3 V) 111(5 V)	50
T_{pd}/ns	3.5	5(3.3 V),7.5(5 V)	3.5(3.3 V),7.5(5 V)	7.5
封装	44～432pin PLCC, TQFP,PQFT,BGA	28-PLCC 28-SSOP	100～272-TQFP PQFP,BGA	20,24,28-PDIP PLCC

7.3.1 ispGDS和ispGDX系列器件

Lattice公司生产的ispGDS和ispGDX,是一种用ISP技术来定义互联关系的数字开关和数字互联器件。ISP技术不仅适用于重构电路的逻辑,还可适用于重构电路的互联关系。

GDS的全称是通用数字开关(Generic Digital Switch),包括ispGDS22,ispGDS18和ispGDS14等3个品种。这些型号尾部数字表示该GDS器件中可供互联用的端口总数。其主要功能是一个开关矩阵,该开关矩阵具有若干个输入/输出端口,内部结构是由分成行、列两组的可编程开关矩阵和可编程I/O单元构成的。通过编程可将行与列的任意一个端口与另一

个端口连接，使任意一个I/O端口接到高电平、低电平或由开关矩阵送来的信号（同相或反相形式）上。图7.3.1为ispGDS22的内部结构框图。每个I/O单元共有5种组态。

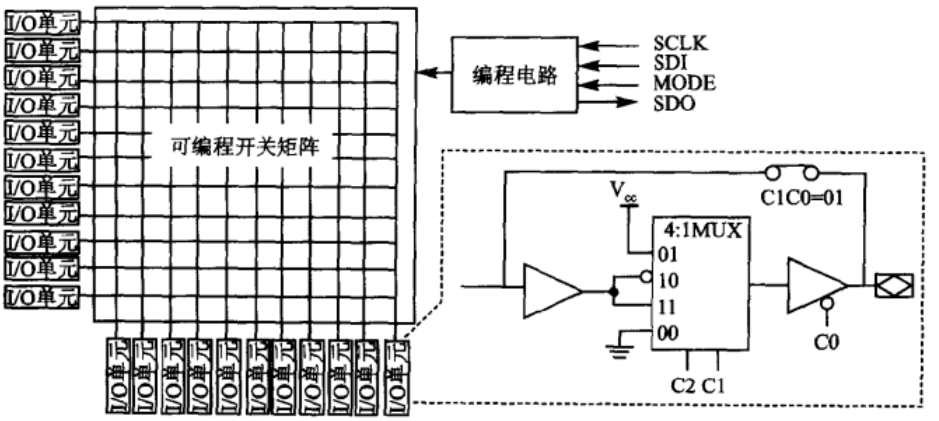


图7.3.1 ispGDS22的内部结构框图

ispGDS器件的最大延时为7.5 ns，工作电压为5 V，电流为40 mA。ispGDS的I/O单元本身有上拉电阻存在，因而用ispGDS器件完全可替代DIP开关。使用ispGDS的最大意义在于：在不拨动机械开关或不改变系统硬件的情况下，能快速地改变或重构印制电路板的连接关系，实现对目标系统连接关系的重构和高性能地完成信号分配与布线，使系统升级更容易，性能选择、系统创建更为简单。

ispGDX是一种在系统编程的通用数字交叉阵列（ISP Generic Digital Crosspoint）器件系列，提供了高集成度、系统级的信号布线和接口。该器件把在系统编程能力带到了印刷电路板（PCB），能在电路板级提供灵活有效的信号走线，使总线互联部件集成化。

ispGDX在结构上由两大模块组成：一个全局布线区（GRP）和环绕四周的可编程I/O单元。通过在系统编程，ispGDX器件的每个I/O引脚连接到高电平或低电平或某路信号线，用以模拟PCB板上的DIP开关和跳线器。它特别适用于诸如多微处理器接口、多位的数据/地址总线以及PCB板信号布线等系统级硬件设计中。

Lattice公司的ispGDX器件有以下几种型号：ispGDX80A/VA，ispGDX120A，ispGDX160A/VA和ispGDX240VA。其I/O端口数分别为80，120，160和240。ispGDXVA类型的器件可由3.3 V电源供电。ispGDX器件的主要性能如下：

- 3.3 V内核电压，可编程3.3 V或2.5 V输出电压选择（ispGDXVA系列）；
- 5 V内核电压（ispGDX系列）；
- 频率：111 MHz（ispGDX系列）、250 MHz（ispGDXVA系列）；
- 引脚间延时：3 ns（ispGDXVA系列）、5 ns（ispGDX系列）；
- 布线满足任意输入至任意输出（Any input to any output）；
- 高速性能；
- 支持4:1～16:1（ispGDXVA系列）多路选择技术；
- 具有边界扫描测试能力；
- 可编程上拉（Pull－up）、总线锁存（Hold Latch）（ispGDXVA系列）、I/O引脚漏极开路技术；

● 易于使用的 ispGDX 开发系统软件(ispGDX™ Development System)。

ispGDS 和 ispGDX 系列器件由相应的开发系统软件进行编程,起到重构电路的作用。首先由编辑器编辑描述设计内容的 ispGDS 源文件(.GDS 文件)或 ispGDX 源文件(.GDX)文件,通过器件适配和编译,生成 JEDEC 文件(.JED 文件),最后可以用 Lattice ISP 菊花链下载软件(ispDownload 工具软件)直接下载到器件中,即将 *JED 文件写入 ispGDS 或 ispGDX 器件。

7.3.2　ispLSI 系列器件

ispLSI(ISP Large Scale Integration)系列器件是 Lattice 公司于 20 世纪 90 年代推出的高性能、高密度可编程逻辑器件,提供 32～840 个宏单元和 32～312 个 I/O 的可编程解决方案。每一个系列针对某一种特定的设计需求,并且能执行多种逻辑功能,包括寄存器、计数器、多路选择器、解码器和复杂状态机。系统工作速度最高可达 225 MHz,而集成密度可达大约 60 000 门。适合在计算机、仪器仪表、通信设备、DPS 系统、雷达和遥测系统中使用。

目前,Lattice 公司的 ispLSI 器件分为 6 个系列。这 6 个系列器件的特点如表 7.3.2 所列。其基本结构和功能相似,都具有在系统可编程能力,但各系列器件在用途上有一定的侧重点,适用对象具有一定的针对性。

表 7.3.2　ispLSI 系列器件特点

系　列	1000	2000	3000	5000	6000	8000
集成门数	2 000～8 000	1 000～6 000	7 000～20 000	12 500～25 000	25 000	25 000～60 000
F_{max}/MHz	60～200	165～225	90～125	100～125	50～70	110
T_{pd}/ns	4.5～20	3.5～5	7.5～12	7.5～12	15～20	8.5
宏单元数	64～192	160～448	256～512	192	192	480～840
寄存器数	96～288	320～672	256～512	416	416	720～1 152
I/O 端口	36～110	35～138	160～224	144～389	159	165～329
封装形式	PLCC,TQFP PQFP,CPGA JLCC	PLCC,TQFP PQFP,CPGA JLCC,HQFP	MQPF,PQFP BGA,CPGA HQFP	BGA	HQFP	BGA

ispLSI1000/E 系列器件是通用器件,集成度较高,性能价格比高,适合在一般的数字系统中使用。

ispLSI2000/E/V/VE 系列器件是高性能、高速 I/O 系统,其中 ispLSI2000E 系列具有目前工业界最快的 3.5 ns(225 MHz)的性能。适合在速度要求高和需要较多 I/O 引脚的电路或系统中使用。例如,移动通信、微处理器接口和高速 PCM 遥测系统等。

ispLSI3000 系列器件集成度高、速度快,适用于数字信号处理、数据加密或压缩等高集成度设计,可把数字系统中的大多数功能集成在一块芯片中,实现"片上系统"。

ispLSI5000V 系列器件是所有 CPLD 产品中输入/输出端口数和乘积项最多的。适合于快速计数器、状态机和地址译码器等。

ispLSI6000 系列器件在结构上增加了存储器。它把存储模块和可编程逻辑电路集成到

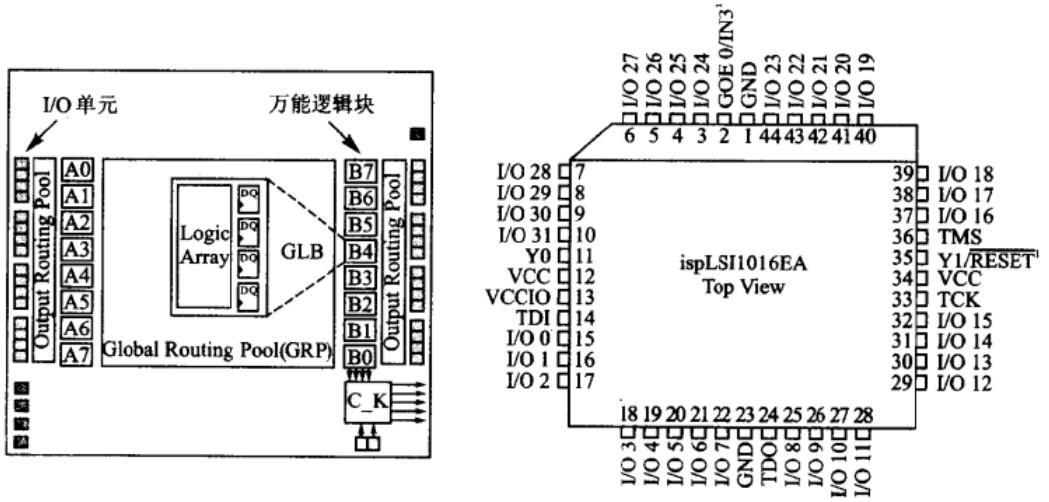
同一块芯片上，是专门为 DSP 等用途设计的芯片。在实现 DSP 功能时，ispLSI6000 器件不需要外接存储器，该存储器被集成在芯片中，因而减少互联延时，提高工作速度，并且还具有边界扫描测试能力。

ispLSI6000/V 系列器件是超大规模集成电路，片内可达到 58 000 个逻辑门的规模。为目前复杂数字系统设计提供了相应的资源和性能。ispLSI 8000/V 超大器件有 5 V 和 3.3 V 型，包括柔性 I/O 引脚。5 V 器件能够与 5.0 V 和 3.3 V 逻辑级兼容。3.3 V 器件能够与 3.3 V 和 2.5 V 级兼容，也具有边界扫描测试能力。

7.4 Lattice 公司的 ispLSI1000 系列器件结构

ispLSI1000 系列器件是较早的成熟的 ispLSI 器件，性能稳定可靠。它的集成度在 2 000～8 000 门之间，系统工作频率可达到 200 MHz，I/O 资源丰富，有 44～128 引脚封装可供选择。ispLSI1000 系列器件共有 4 个型号：ispLSI1016，ispLSI1024，ispLSI1032 和 ispLSI1048。ispLSI1000E 系列是 ispLSI1000 系列的改进型，其结构相似，只是 ispLSI1000E 系列器件增加了两个新的输出使能引脚，并且可以控制输出信号的翻转速度。下面以 ispLSI1016EA 为例，说明 ispLSI 可编程逻辑器件系列的结构及功能特点。

图 7.4.1 是 ispLSI1016EA 结构框图和引脚图(PLCC 封装)。ispLSI1016EA 有 32 个I/O 单元，每个单元对应一个 I/O 引脚。从图中可以看出，该器件结构可分为五部分：集总布线区、万能逻辑块、输入/输出单元、输出布线区和时钟分配网络。



(a) 结构框图　　(b) 引脚图 （PLCC ）

图 7.4.1　ispLSI1016EA

1. 集总布线区

集总布线区 GRP(Global Routing Pool)位于芯片的中央。它是器件的内部连线资源，可将所有片内逻辑联系在一起。GRP 可提供 100 %的连线布通率，而且其输入/输出之间的延时是恒定和可预知的。例如 110 MHz 档级的芯片在带有 4 个 GLB 负载时其延时时间为 0.8 ns，与输入、输出的位置无关。这使片内互联性能臻为完善，使用者可以很方便地实现各

种复杂的设计。

2. 万能逻辑块

万能逻辑块 GLB(Generic Logic Block)是位于集总布线区 GRP 四周(或两边)的小方块，每边 8 块，ispLSI1016 器件共 16 块(A0～A7，B0～B7)。它是 ispLSI 器件中最基本的逻辑单元，能够实现主要的逻辑功能。GLB 也是"与-或"阵列结构。它有很多根输入信号线，能够实现复杂的逻辑函数。图 7.4.2 是 GLB 的功能框图。它由四部分组成，即与阵列、乘积项共享阵列、输出逻辑宏单元和控制逻辑。

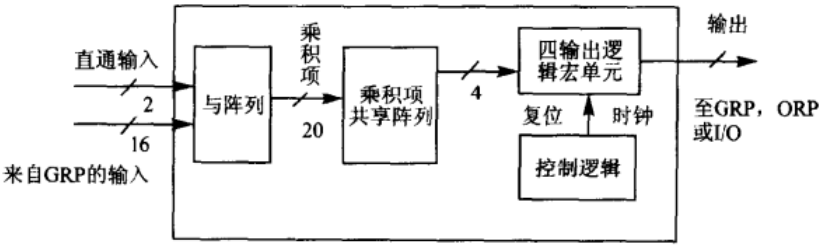


图 7.4.2　GLB 功能框图

1016 的与阵列有 18 个输入线和 20 个乘积项输出线。18 个输入线中有 16 个来自集总布线区，其余两个信号来自专用 I/O 直通输入，输入信号通过互补缓冲器进入与阵列，并且可提供输入的原变量和反变量；每个乘积项的输出是 18 个输入信号的任意组合形式的与函数，再通过 4 个或门输出，如图 7.4.3 所示。

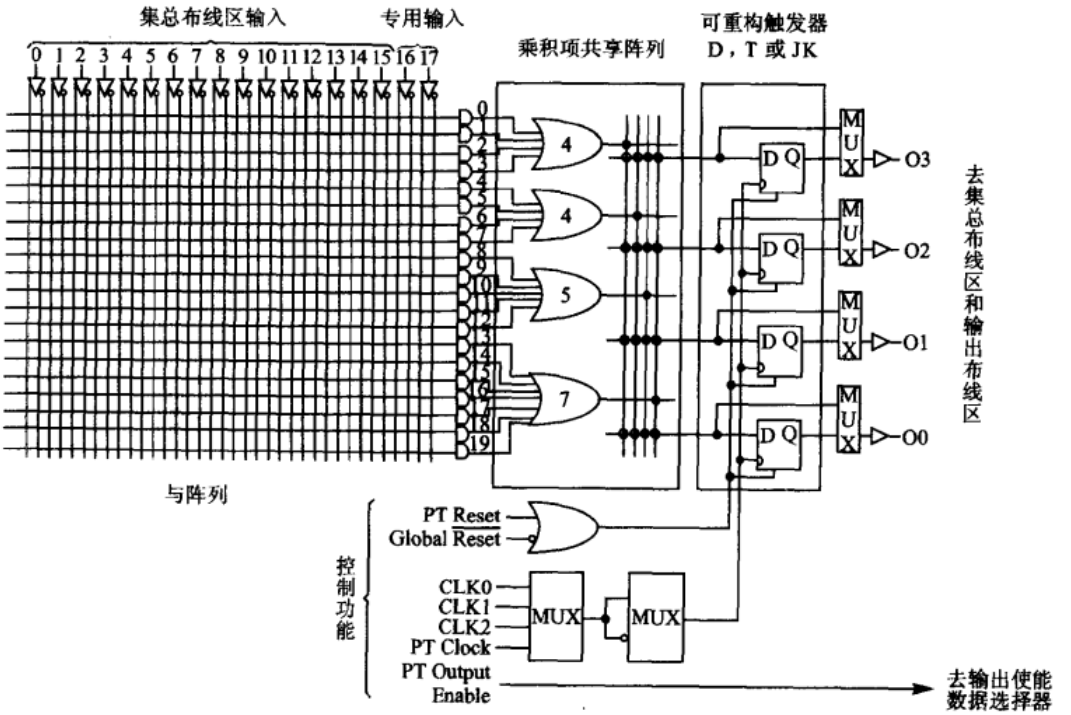


图 7.4.3　万能逻辑块的内部结构(标准配置)

输出逻辑宏单元中有 4 个触发器，每个触发器与其他可组态电路间的连接类似 GAL 的 OLMC(图 7.4.3 中未画出)。它可被组态为组合输出或寄存器输出(靠触发器后面的 MUX 编程组态)，组合电路可有“与或”或者“异或”两种方式，触发器也可组态为 D,T 或 JK 等形式。

乘积项共享阵列 PTSA(Product Term Sharing Array)是 GLB 中的一个特殊结构，ispLSI 器件的逻辑功能体现在乘积项共享阵列灵活的配置上。从图 7.4.4 可以看到，乘积项共享阵列的输入来自 4 个或门，而其 4 个输出则用来控制该单元中的 4 个触发器。至于哪一个或门送给哪一个触发器不是固定的，而是靠编程决定。一个或门输出可以同时送给几个触发器，一个触发器也可同时接受几个或门输出的信息；有时为了提高速度，还可以跨过 PTSA 直接将或门输出送至某个触发器。GRP 输出的 20 个乘积项按 4,4,5,7 分配给这 4 个或门，每个或门输入最上面一个乘积项(号码为 0,4,8,13)可以通过编程从对应的或门中游离出来，而跟或门的输出构成异或逻辑，乘积项中 12,17,18,19 也可不加入相应的或门。此时可见，由于 PTSA 的存在，使得 1016 器件在乘积项共享方面灵活得多。

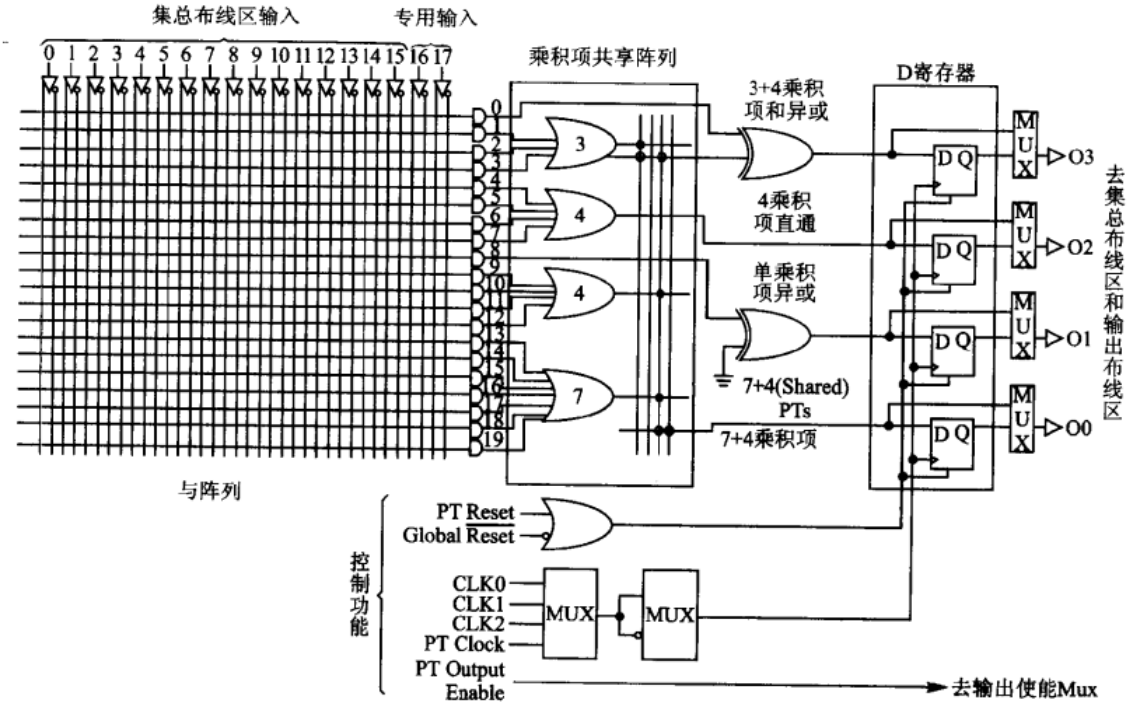


图 7.4.4 GLB 的多模式配置

为了满足设计需要，乘积项共享阵列有五种配置模式：标准配置、高速旁路配置、异或配置、单乘积项配置或者多模式配置。多模式配置是其余各模式在同一个 GLB 中混合使用构成的多模式组态，如图 7.4.4 所示。其中输出 O3 采用的是(3+4)乘积项驱动的异或模式，实现异或函数；O2 采用的是高速旁路配置的 4 乘积项直通模式；O1 采用的是高速旁路配置的单乘积项模式，为第 8 乘积项提供旁路；O0 采用的是(7+4)乘积项驱动的标准模式。

输出逻辑宏单元中 4 个 D 触发器的时钟是连在一起的，同一 GLB 中的触发器必须同步工作，且所使用的时钟信号可以有多种选择，可以是全局时钟，也可以是片内生成的乘积项时钟。在图 7.4.3 控制功能部分的两处 MUX 中，左边一个用来选择时钟信号；右边一个用来控

制时钟极性。4 个时钟信号中 CK0,CK1 和 CK2 由芯片内的时钟分配网络提供,乘积项时钟则由乘积项 12 产生。正因为有此选择功能,所以不同 GLB 中触发器可以使用不同的时钟。

同样,4 个 D 触发器的复位端也是相连在一起的,复位信号可以是全局复位信号(Global reset)或者本 GLB 中乘积项 12,19 产生的复位信号,两者始终是或的关系,这样在 GLB 内,4 个触发器同时复位,而不同 GLB 之间则可以不同时复位。

GLB 每个输出对应的三态门(实际存在于 IOC 中)的使能信号,如果需要也由本 GLB 的乘积 19 提供。当然,乘积项 12,19 作复位、时钟或输出使能用时便不能再作为逻辑项使用。

因此万能逻辑块 GLB 是 ispLSI 芯片中最关键的一种标准逻辑块,ispLSI1000,2000 系列的 GLB 都与此相同,ispLSI3000,6000 系列的 GLB 则采用了孪生 GLB(Twin GLB),即一个 GLB 中有两个这样的逻辑块,共有 24 输入,每个逻辑块各自产生 20 个乘积项,最终可以获得两套四端口输出。

3. 输入/输出单元

输入/输出单元 IOC(Input Output Cell)是图 7.4.1(a)中最外层着色的小方块,共有 32 个。其结构如图 7.4.5 所示。该单元有输入、输出和双向 I/O 三类组态,靠控制输出三态缓冲电路使能端的 MUX 来选择。该 MUX 有两个可编程的地址,图中所画为未编程状态,此时二地址输入端皆接地,相应于 00 码,因而将高电平接至输出使能端,IOC 处于专用输出组态;若两地址输入中有一个与地的连接断开,即地址码为 10 或 01,则将由 GLB 产生的输出使能信号来控制输出使能,处于双向 I/O 组态或具有三态缓冲电路的输出组态;若两地址与地连接皆断开,则将输出使能接地,处于专用输入组态。

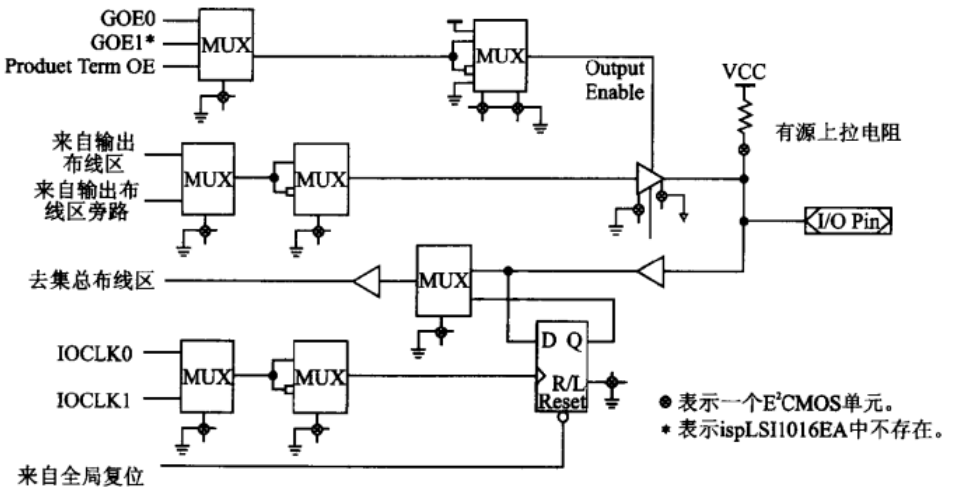


图 7.4.5　IOC 结构图

图 7.4.5 中第二行两个 MUX 用来选择输出极性和选择信号输出途径。第三行的 MUX 则用来选择输入组态时用何种方式输入。IOC 中的触发器是特殊的触发器,可以用两种方式工作:一是锁存(Latch)方式,触发器在时钟信号"1"电平时透明,"0"电平时锁存;二是寄存器(Register)方式,在时钟信号上升沿时将输入信号存入寄存器。这两种方式靠对触发器的R/L端编程确定。触发器的时钟也由时钟分配网络提供,并可通过第四行的两个 MUX 选择和调整极性。触发器的复位则由芯片全局复位信号 Reset 实现。

综合上面各功能可以得到图 7.4.6 所示的各种 I/O 组态，再与各 GLB 组态以及对 GLB 中 4 输出宏单元的组态方式相组合，便可得到不下几十种电路方式。所以 ispLSI 是非常灵活的，可以和 FPGA(现场可编程器件)相比拟。每个 I/O 单元还有一个有源上拉电阻，当该 I/O 端不使用时，该电阻自动接上，可以避免因输入悬空引入的噪声和减小电路的电源电流。正常工作时如接上上拉电阻也具有此优点。

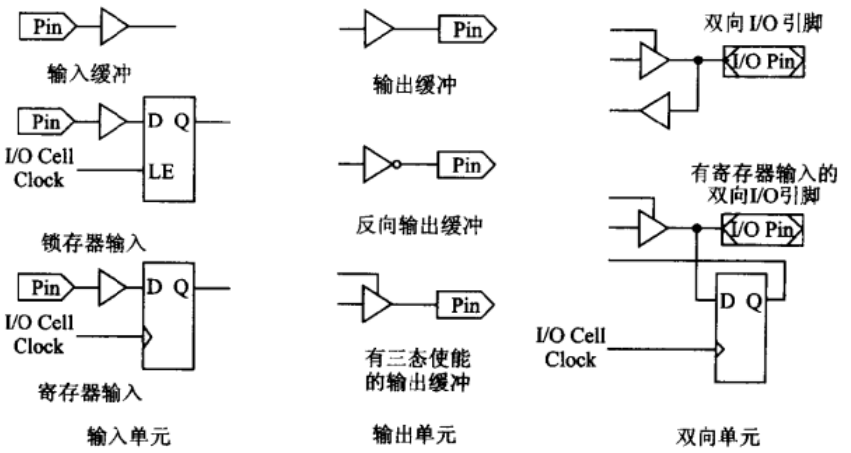


图 7.4.6 IOC 组态举例

4. 输出布线区

输出布线区 ORP(Output Routing Pool)是 ispLSI 器件独有的结构，负责 GLB 输出信号到 IOC 的连接。利用 ORP 能够改变 GLB 输出线到引脚之间的连接，增加引脚分配的灵活性。图 7.4.7 是 ORP 的逻辑图。ORP 是介于 GLB 和 IOC 之间的可编程互联阵列，阵列的输入是 8 个 GLB 的 32 个输出端，阵列有 16 个输出端，分别与该侧的 16 个 IOC 相连。通过对 ORP 的编程，可以将任一个 GLB 输出灵活地送到 16 个 I/O 端的某一个。

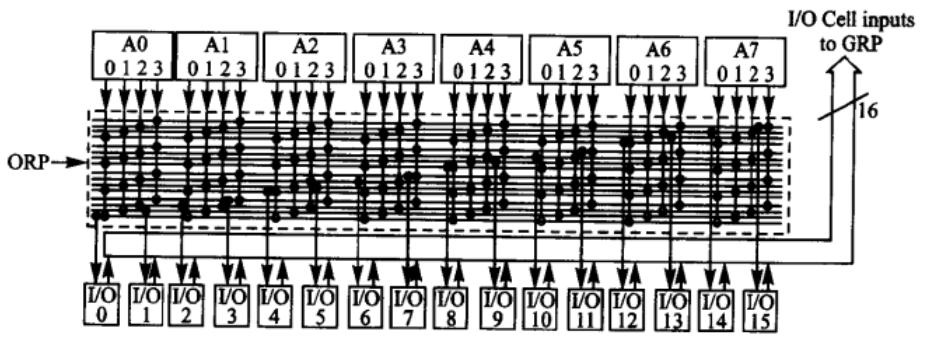


图 7.4.7 ORP 逻辑图

可见 1016 的一大特点是 IOC 与 GLB 之间没有一一对应的关系，因而可以将对 GLB 的编程和对外部引脚的排列分开进行，并可实现在不改变外部引脚排列的情况下修改芯片内部的逻辑设计。

在 ORP 旁边还有 16 条通向 GRP 的总线，I/O 单元可以使用，GLB 的输出也可以通过 ORP 使用，从而方便地实现了 I/O 端复用的功能和 GLB 之间的互联。

为了进一步提高器件的速度和灵活性，使用 ORP 旁路配置时，GLB 的输出还可跨过 ORP 直接与 I/O 单元相连，如图 7.4.8 所示。旁路 ORP 可以减少系统延时，但会限制器件的布线能力，只有关键信号才可使用 ORP 旁路连接。

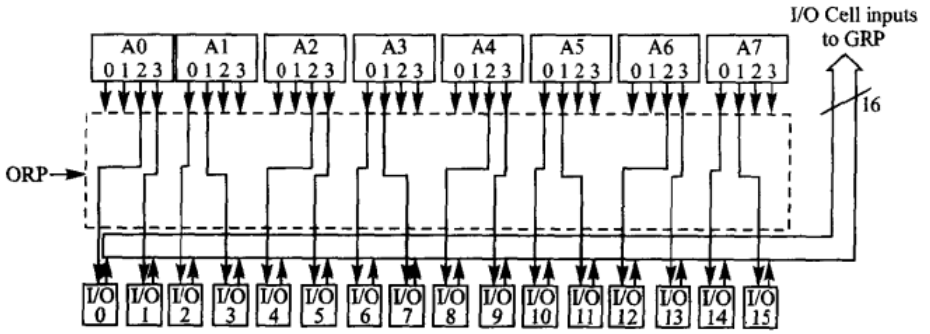


图 7.4.8　跨越 ORP 连接方式

5. 时钟分配网络

时钟分配网络 CDN(Clock Distribution Network)在图 7.4.1(a)的右下角，逻辑结构图如图 7.4.9 所示。它产生的全局时钟信号有 5 个，其中 CLK0，CLK1，CLK2 为提供给所有 GLB 的时钟；IOCLKI0 和 IOCLK1 提供给所有 I/O 单元作为时钟。

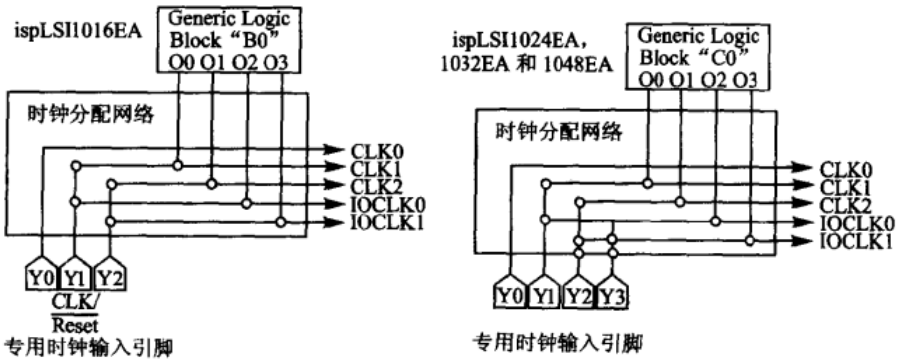


图 7.4.9　时钟分配网络逻辑结构图

ispLSI1000 系列器件的时钟分配网络输入信号由 4 个专用输入端 Y0，Y1，Y2 和 Y4 提供，这 4 个引脚上的时钟可以分配给任意 GLB 和 IOC。但 ispLSI1016 器件只有 3 个专用输入端 Y0，Y1，Y2，而其中 Y1 兼有时钟或复位的功能。除此之外，还可以将时钟专用 GLB"C0"(在 1016 中是 B0 单元)的 4 个输出送入 CDN，以建立用户定义的内部时钟电路。例如将外加主时钟由 Y0 送入作为全局时钟 CLK0，此全局时钟通过时钟专用 GLB"C0"(在 1016 中是 GLB"B0"单元)分频后送至 CLK0，CLK1，CLK2，IOCLK0，IOCLK1，则其他 GLB 或 I/O 单元便可以在比外部输入主时钟频率低的节拍上工作。

6. 宏模块结构

在 ispLSI1000 系列中采用了一种大的分块结构，即宏模块(Megablock)结构。每 8 个 GLB 或 4 个孪生 GLB，连同对应的 ORP，IOC 等构成一个大块，每个 ispLSI 器件都由若干大块构成，1016 共有两个大块。图 7.4.10 是 1016 的大块方框图，其中除了上述部分外，还包括

两个专用输入端和一个公共的乘积项 OE，这两个输入端是不经过锁存器直接输入的，且只能为本大块内的 GLB 使用，靠软件自动地分配。乘积项 OE 由该大块中某个 GLB 的 19 号乘积项产生，以作为该大块所有 16 个 I/O 单元公用的 OE 信号（对于工作在三态输出模式的 I/O 单元而言），从而避免了每个需三态输出的 GLB 皆要产生 OE 信号的弊端，而各 I/O 单元 OE 端与该乘积项的连接是靠一个 8 选 1 OE MUX 实现的，如图 7.4.11 所示。

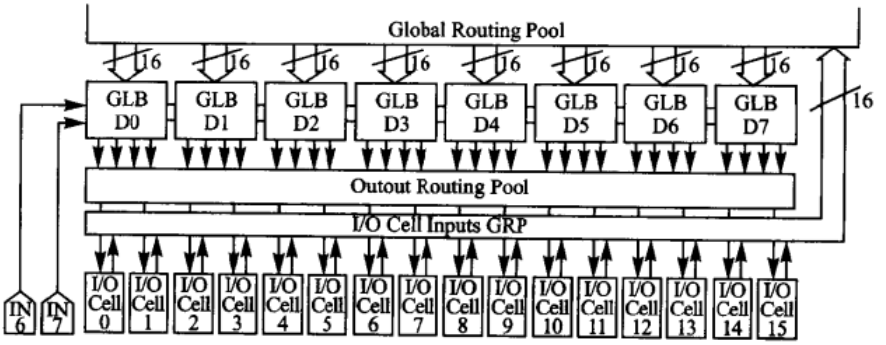
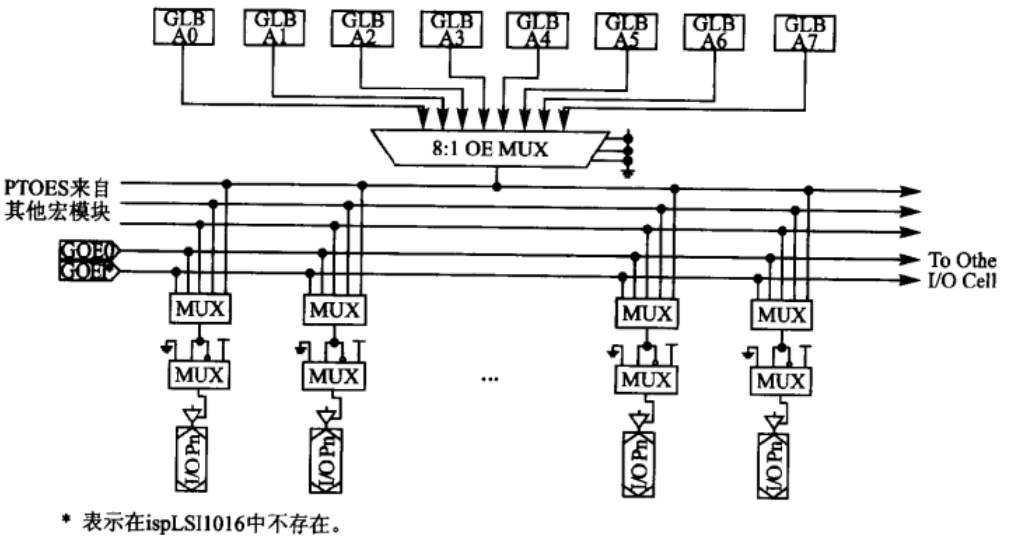


图 7.4.10　1016 的宏模块方框图



* 表示在ispLSI1016中不存在。

图 7.4.11　宏模块的输出使能控制

第八章　可编程逻辑器件的设计与开发

8.1　可编程逻辑器件的设计流程

可编程逻辑器件无论是熔丝型、UV 可擦型还是电可擦型，其编程过程都是在计算机控制下，按照一定的程序给芯片需要编程处理的地方加上某种特殊波形的电流脉冲完成，但哪些地方该加或不该加，则需根据设计要求生成的编程数据文件确定。通常在编程下载操作前需要进行设计输入、设计实现和设计仿真等工作。可编程逻辑器件设计流程如图 8.1.1 所示。

1. 设计输入

设计输入就是根据系统设计要求对所设计的任务提出一个简洁而完整的功能描述，并且以开发软件要求的某种输入形式表示。常用设计输入有电路原理图、硬件描述语言和波形输入等形式。

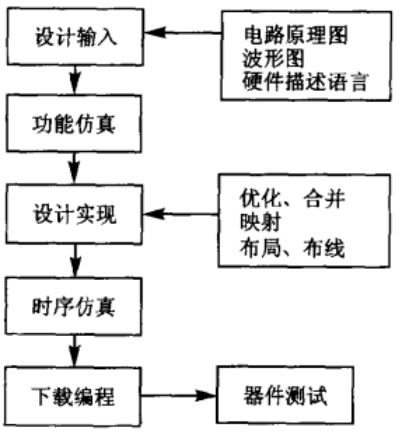


图 8.1.1　可编程逻辑器件设计流程

电路原理图是图形化的表达方式，使用元件符号和连线来描述设计。其特点是比较容易掌握，直观而方便，所画的电路原理图与传统的器件连接方式完全一样，不需要重新学习，很容易被人接受。尤其对系统的层次结构、模块化结构更为方便，适合于描述系统逻辑的连接关系和接口关系。但它要求开发软件设计工具提供必要的元件库或逻辑宏单元以及用户自己建立的元件。如果设计规模大、连线复杂，则设计的易读性差，很难搞清电路的实际逻辑功能。采用原理图输入方式较不方便，就需要采用硬件描述语言输入。如果从输入、输出的逻辑关系描述，还可以采用波形激励方式输入，这样可从信号一级把握输入。

硬件描述语言是采用文本编程的方式描述设计，其逻辑描述功能强。一般分为普通硬件描述语言和行为硬件描述语言两种。普通硬件描述语言，如 AHDL，ABEL－HDL，CUPL 和 MINC－HDL 等，支持真值表、布尔方程（逻辑函数）、状态机等逻辑描述方式，适合对计数器、译码/编码器、比较器和状态机等逻辑功能的设计；行为硬件描述语言，如 VHDL，类似于高级语言，并且还是一种并行语言。在描述复杂设计时，它非常简洁，具有很强的逻辑描述和仿真功能，已成为国际标准。

可编程逻辑器件的设计多采用层次化的设计方法，分模块、分层次地进行设计描述。描述器件总功能的模块放置在最上层，称为顶层设计；描述器件最基本功能的模块放置在最下层，称为底层设计。顶层和底层之间类似于程序中的主程序和子程序的关系。一般顶层设计中，使用描述逻辑连接和接口关系的原理图输入方式；在底层设计中，使用硬件描述语言描述逻辑功能。每个功能模块可使用原理图输入或者硬件描述语言描述，以及原理图和硬件描述语言

相结合的输入。设计者利用硬件描述语言创建一个逻辑关系，同时以图形符号形式表达出来。这便是图形和行为逻辑设计结合的输入方式。

2. 设计实现

设计实现是从输入设计文件到生成下载数据文件(熔丝图文件或位流文件)的编译过程。此部分是开发软件工具的核心部分，虽然软件本身给用户提供了一些控制操作，但大部分工作都由开发软件自动完成，所以 EDA 的厂商都力求使编译过程到达完全智能化。设计实现主要完成以下 4 个相关任务：

① 优化和合并：优化是指逻辑化简，把逻辑描述转变为最适合在选定器件中实现的形式。合并是将模块化设计产生的多个文件合并为一个网表文件，并使层次设计平面化。

② 映射：是把设计分为多个适合器件内部逻辑资源实现的逻辑小块的形式。如 Lattice 公司的 ispLSI 系列器件中，适合用 GLB 和 IOC 来实现；在 Altera 公司的 MAX，FILE 系列中，适合用 LAB 和 IOE 来实现。

③ 布局和布线：布局是将已分割的逻辑小块放到器件内部逻辑资源的具体位置，并使它们易于连线，且连线最少；布线是利用器件的布线资源完成各功能块之间和反馈信号的连接。器件连线、资源布局及设计的复杂程度都将影响布线的成功率，即布通率。另外，布局上的问题也会引起布线困难。这就需要重新修改设计输入或改变设计策略来解决布线问题。

④ 生成编程文件：设计实现的最后一步是产生可供器件编程使用的数据文件。对 CPLD 器件产生熔丝图文件，即 JEDEC 文件；对于 FPGA 器件则产生位流数据文件 Bitstream。

3. 设计仿真

这部分的最大功能是便于用户检查自己的设计思想是否得到实现和设计中存在的问题。可以在设计过程中对整个系统乃至各个模块进行近似实际的软仿真，即在计算机上用软件验证连接和逻辑功能是否正确，各部分的时序配合是否准确。如果有错可以方便地修改错误，而不必在硬件上做改动。高级的仿真软件还可以对整个系统设计的性能进行评估。规模越大的系统，越发需要设计仿真，最后才能正确地将设计付诸电路甚至板级实现。

设计仿真包括前仿真、后仿真和器件测试三个部分。在设计输入后进行的功能仿真是检验设计的逻辑功能，又称为前仿真；设计实现后的时序仿真又称后仿真，是在选择了具体器件并完成了布局、布线后进行的定时关系仿真。由于不同器件的内部延时不一致，不同的布局、布线方案也给延时造成很大的影响，因此在设计实现后，对设计和逻辑块进行延时仿真、分析定时关系、估计性能是非常有必要的。在器件编程后，需要利用检测手段测试器件最终的功能和性能指标。具有边界扫描测试能力和在系统编程能力的器件，测试起来较其他器件方便。

4. 下载编程

下载编程是将设计输入通过编译生成的 JEDEC 文件或位流文件装入到可编程器件中。器件编程需要满足一定的条件，如编程电压、编程时序和编程算法等。普通的 CPLD 器件和一些 FPGA 器件需要专用编程设备完成器件编程。基于 SRAM 的 FPGA 器件可以由 PROM、微处理器或并行(串行)下载电缆下载。在系统编程器件不需要使用编程器，只是简单利用 PC 机的并口和简单的编程电缆进行编程下载。

8.2　MAX+PLUS II 软件开发系统

8.2.1　MAX+PLUS II 开发工具简介

Altera 公司的开发工具已经历了四代，从最初的基于 DOS 的 A+PLUS，发展到 MAX+PLUS，又于 1991 年推出性能更加完善的基于 Windows 的开发工具 MAX+PLUS II。目前该公司又推出了它的第四代开发工具 Quartus II，提供了为可编程片上系统 SOPC(System-On-a-Programmable-Chip)设计的开发环境。随着器件结构、性能的不断发展和集成度的不断提高，Altera 公司始终能够同步推出与之相适应的开发工具，以满足设计需要。

1. 安装 MAX+PLUS II 系统

MAX+PLUS II 系统具有容易学、容易使用的特点，可以安装在任何操作系统下。MAX+PLUS II 软件有两种版本：商业版和开放版。

商业版：可以进行功能分析、时序分析、各种文本及图形输入、下载其厂家的各种芯片；附加一个软件狗。

开放版：在商业版上加以限制，是免费的。只有向 Altera 公司申请一个授权码或 license.dot 文件，才可以使用。

(1) 系统配置要求

① 486 或奔腾以上 CPU；

② Windows NT 3.5 或 Windows 95 以上版本；

③ 光盘驱动器、双键鼠标及并行口；

④ MAX+PLUS II 至少需要 120 MB 的硬盘空间，建议用 1.2 GB 以上的硬盘；

⑤ 不同器件编译所需要的内存大小各不相同：32 MB(MAX7000)，56 MB(FILE10K)。

(2) 安装步骤

① 将光盘放进光驱，进入 Windows 界面的资源管理器中，选择光驱中\PC\MAX+PLUS2\INSTALL.EXE 文件。

② 双击 INSTALL.EXE 文件，进入 Install 界面。根据界面要求，一步一步往下运行即可。首次运行 MAX+PLUS II，会出现“License Agreement”(授权协议)对话框，需要先到指定的 Internet 站点上申请授权号，填入到“Authorization Code”对话框中，单击“OK”按钮，即可使用。

2. MAX+PLUS II 软件特点

MAX+PLUS II 全称为 Multiple Array Matrix and Programmable Logic User System。当前 MAX+PLUS II 是市场上使用最广的开发工具软件之一，是一个功能强、使用方便的设计工具。MAX+PLUS II 软件提供了一种与结构无关的设计环境，设计者无需精通器件内部的复杂结构，而只需要使用常用的设计输入方法(如原理图输入、HDL 和波形输入)进行描述，MAX+PLUS II 软件会自动在计算机上把设计输入编译成最终结构所需要的格式。MAX+PLUS II 开发软件具有如下特点：

① 结构无关。MAX+PLUS II/Compiler(编译程序)是该开发软件系统的核心，能自动完成优化和逻辑综合。它支持 Altera 公司的 Classic，MAX 和 FILE 可编程器件系列，提供了

工业界中惟一真正与结构无关的可编程逻辑设计环境。

② 多平台。MAX+PLUS II 是将输入编辑、编译和校验功能全集成化的一套可编程开发工具，可以加快动态调试，缩短设计周期。

③ 硬件描述语言。支持各种 HDL 设计输入形式，包括 VHDL，Verilog HDL 和 Altera 公司的硬件描述语言 AHDL。

④ 开放的界面。MAX+PLUS II 提供了与其他工业标准设计输入、综合和校验工具的链接。它与 CAE 工具的接口符合 EDIF200/300，LPM，VHDL，Verilog HDL 及其他标准。目前 MAX+PLUS II 支持主流的第三方 EDA 工具，如 Synopsys，Viewlogic，Mentor Graphics，Cadence，Exemplar，Data I/O，Intergraph，Minc，OrCAD 等公司的工具。

3. MAX+PLUS II 的设计流程

使用 MAX+PLUS II 软件开发工具进行设计的整个过程如图 8.2.1 所示，大体上可分为四个阶段：设计输入、设计实现、设计验证和器件编程。设计输入中的输入方式有三种：图形编辑、波形编辑和文本编辑。图形编辑即输入电路原理图，不仅可以使用 MAX+PLUS II 中丰富的图形器件库，而且可以使用几乎全部的标准 EDA 设计工具。文本编辑方式支持 Altera 公司的 AHDL 语言，并且兼容 VHDL 和 Verilog HDL。波形编辑最有特点，它允许设计者通过编辑输入/输出波形的逻辑关系，由系统自动生成该功能模块。此外，符号编辑器用于编辑用户自己的功能模块符号。通过底层编辑器可以观察实际器件的内部结构，并可指定和改变器件引脚分布，或者调整各模块在器件内部宏单元之间的分布，从而优化器件性能。

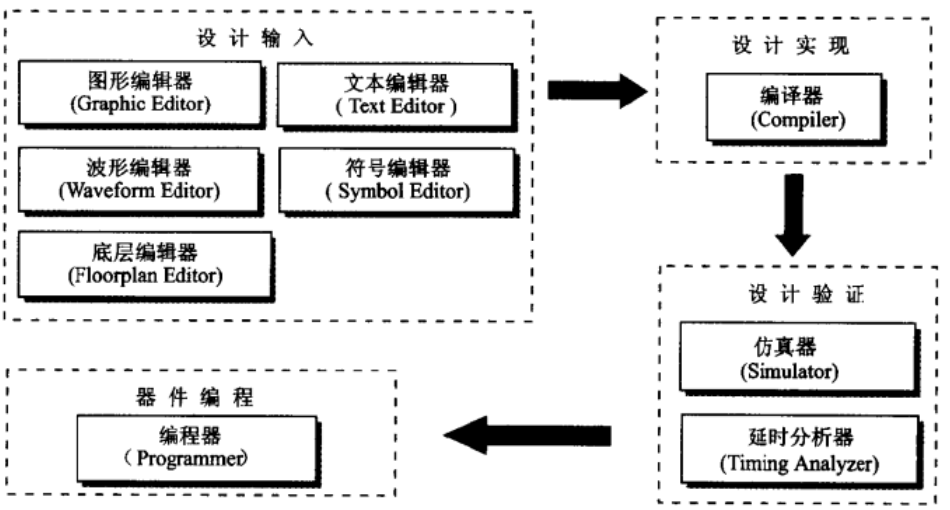


图 8.2.1 MAX+PLUS II 的设计环境

设计实现是在所选的可编程逻辑器件内物理地实现所设计的逻辑。其过程主要由 MAX+PLUS II 中的核心部分编译器(Compiler)完成。编译器是依据设计输入文件和选定的目标器件自动生成用于器件编程、波形仿真及延时分析等所需要的数据文件。

仿真器和延时分析器利用编译器产生的数据文件自动完成逻辑功能和延时特性仿真。在仿真文件中加载不同的激励，可以观察中间结果以及输出波形。在仿真结果正确以后，就可以进行器件编程，即通过编程器(Programmer)将编译的数据文件下载到实际芯片中。下载之后，仍需进行动态仿真，因为在前面的仿真属于静态时序仿真，并未涉及实际器件。动态仿真

是将实际信号送入实际芯片中进行的时序验证。最后测试芯片在系统中的实际运行性能。

整个设计过程中所使用的设计工具都集成在 MAX+PLUS II 之 中，每个工具以窗口形式打开，并且可以同时打开多个窗口，通过在各窗口之间切换，分别使用各个设计工具。

8.2.2　设计输入

根据图 8.2.1 所示，可编程逻辑器件从设计输入到器件编程这四个阶段可在 MAX+PLUS II 软件提供的环境下完成。进入 Windows 后，双击 MAX+PLUS II 图标，进入MAX+PLUS II 主界面，如图 8.2.2 所示。

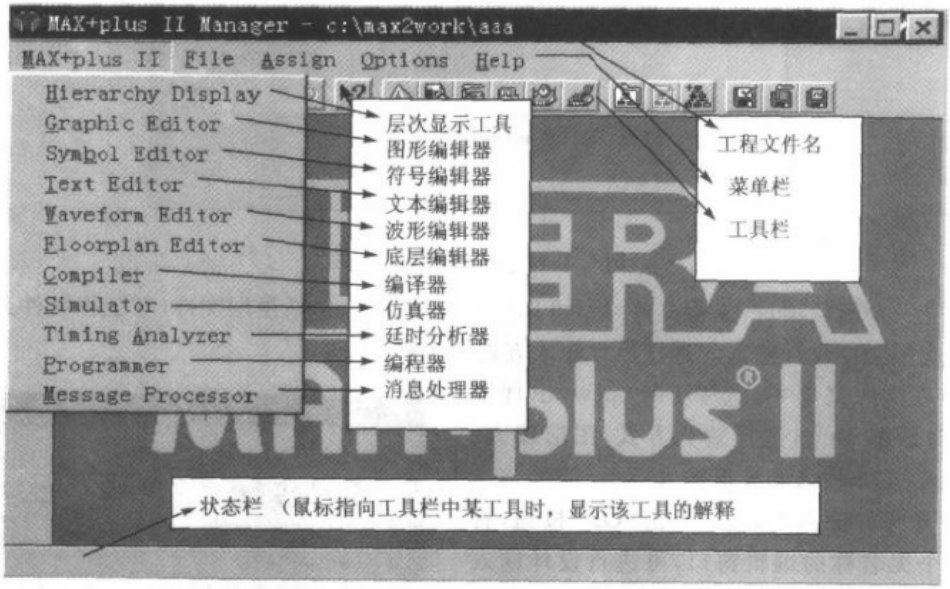


图 8.2.2　MAX+PLUS II 的主界面

1. 建立新项目、确定工程文件名

在开始设计之前，首先应了解“工程”的概念。一个工程(Project)是一个系统设计的总和，包含了所有的子设计文件和设计过程中产生的所有辅助文件。一个系统由多个模块组成，每个模块由多个子系统构成，系统具有多层次结构。对系统设计可按层次的概念来理解，即“自底向上”和“自顶向下”的设计方法，所有子设计文件是底层文件。它们可以是并列关系，也可以是包含关系，层次的深度没有限制。最顶层文件描述了系统的整体关系，代表了自底而上所有设计的总和。顶层文件同子设计文件(底层文件)一样，也可以以图形或文本文件来描述(波形设计文件只能作底层文件类型，不能作为顶层文件)。由于编译器、仿真器等是面向工程文件的，也就是说，编译器编译的是当前的工程文件。要对某个文件进行编译，必须具有与该文件相同的工程文件名。一般最顶层文件名与工程文件名是相同的。建立新项目，确定工程文件名，其方法如下：

① 选菜单 File\Project\Name，显示对话框，工程文件名对话框如图 8.2.3 所示。

② 在 Project Name 框中填入工程文件名(如 aaa)，单击“OK”按钮，则 MAX+PLUS II 窗口标题栏会显示当前的工程文件名，如 c:\max2work\aaa，如图 8.2.2 所示。每次进入 MAX+PLUS II 时，系统自动调入上一次编译的工程文件。还可以通过工具栏中的  ，使当前窗口

的文件名转换为工程文件。

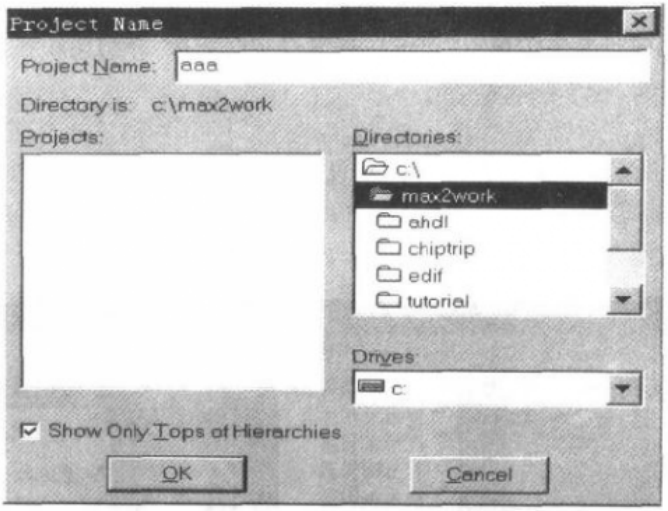


图 8.2.3 工程文件名对话框

建立一个新的项目(工程文件)后,可以进行设计输入,包括顶层文件(与工程文件名相同)或多个底层文件设计输入。选择设计输入方式的步骤如下:

① 在文件菜单中,选中 File\new,出现图 8.2.4 对话框 New,选择输入方式:图形输入、符号输入、文本输入和波形输入。

② 选择好输入方式后,单击"OK"按钮,将弹出一个无名称的编辑窗口,可进行设计输入编辑。

③ 在无名称的编辑窗口中,选 File\save 或 save as,出现 save as 窗口,在 File name 栏中填入输入文件名,然后单击"OK"按钮即可。

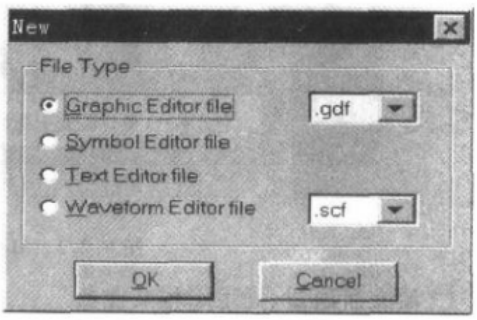


图 8.2.4 New 对话框

2. 原理图输入

图形编辑器能方便快捷地输入设计原理图。该图形编辑器提供了 74 系列的 300 多个元器件以供调用,并且还支持把用文本输入方式设计好的功能块生成一个图形符号的功能。在使用该功能模块时,直接调用该模块的图形符号即可。建立图形输入文件的方法如下。

(1) 选择图形输入方式

在图 8.2.4 所示的对话框中选择 Graphic Editor file(图形编辑文件),单击"OK"按钮,弹出一个无名称的图形编辑窗口,如图 8.2.6 所示。

(2) 输入电路原理图

在图形编辑窗口可进行电路原理图的输入。

① 输入逻辑功能符号:可以选择菜单 Symbol\Enter Symbol... ;更快捷的方法是用鼠标双击图形编辑窗口中的空白处,将弹出一个窗口,如图 8.2.5 所示。在 Symbol name 栏内填入元器件符号名(如 dffe D 触发器),单击"OK"按钮即可。在不知道器件符号名称时,双击相应符号库目录,在符号文件框内选择元器件。用同样的方法依次选中 INPUT,OUTPUT

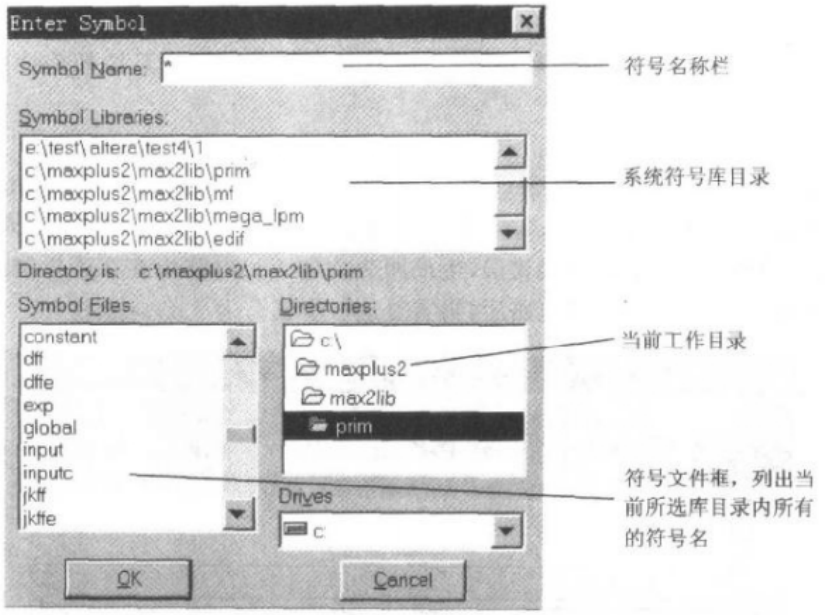


图 8.2.5　Enter Symbol 对话框

和 VCC。

② 复制、移动功能符号：需要复制元器件时，先选中元件符号，此时该符号边缘的虚线变为红色粗实线；然后按住键盘上<Ctrl>键(此时屏幕上鼠标右上方出现一个小加号)，用鼠标左键击中元件符号并拖动，松开鼠标即可。如要删除元件符号，则先选中该符号，用“Delete”键即可删除。若要同时移动多个符号，则先用鼠标左键画一个将所要移动的符号包括在内的大矩形，此时这些元件符号被选中；然后用与前面相同的方法，使多个元件符号同时复制或移动。

③ 引脚的命名：绘制完所有的符号后，应注意到所有的输入/输出引脚名被系统默认为 PIN_NAME 名，用鼠标左键双击“PIN_NAME”，使其变为黑底白字显示，然后可直接填入引脚名。电路原理图如图 8.2.6 所示。

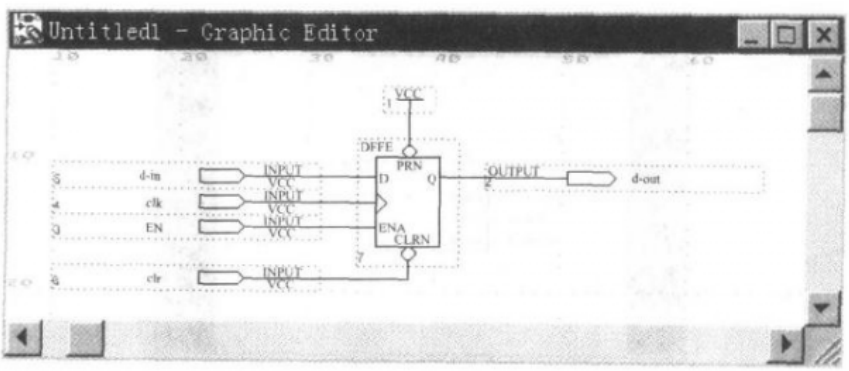


图 8.2.6　电路原理图

④ 连接各符号：同画图的方法一样，将鼠标移至引线端处，鼠标箭头会自动变成十字形状。此时按住左键拖动至另一个引线端处，松开鼠标，连接线即可完成。图 8.2.6 为一个 D

触发器电路原理图。

主窗口左边工具盘内的绘制直线、弧线等工具是为绘制标题或其他装饰性图形、文字用的，绘制电路图时不用这些工具。

(3) 保存文件并检查错误

选中菜单 File\Project\Project Save &Check，系统会自动弹出编译器窗口(用菜单MAX+PLUS II\Compiler 也可以)，如图 8.2.7 所示，生成网表文件(*.cnf)。这步是编译过程的第一步，主要是查找电路中的逻辑连接错误，生成网表文件。如电路中存在逻辑错误，则系统会弹出信息处理窗口，根据信息修改电路后，再重复这一步骤。

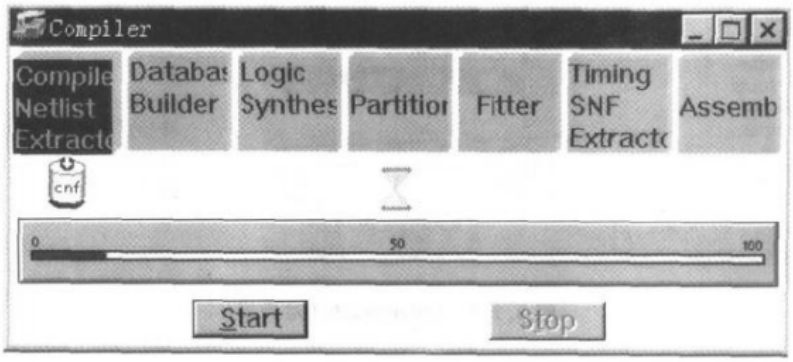


图 8.2.7 编译器窗口

(4) 创建功能模块符号

为了在顶层文件设计中绘制底层文件或子系统描述的电路，常把底层文件或子系统描述的电路封装成一个功能模块，用一个符号来代替。这正是层次设计的关键，即通过创建代表底层设计文件的符号，使得顶层设计文件能够包含所有子设计文件，并且设计结构简洁清晰，逻辑关系明确。

选中菜单项 File\Create Default Symbol，系统自动对当前编辑器中的内容(不仅仅是电路图)生成一个默认符号，如图 8.2.8 所示。此时在文件管理器中可看到生成了一个新的文件(*.sym)。创建好的模块符号可以被其他设计输入调用。

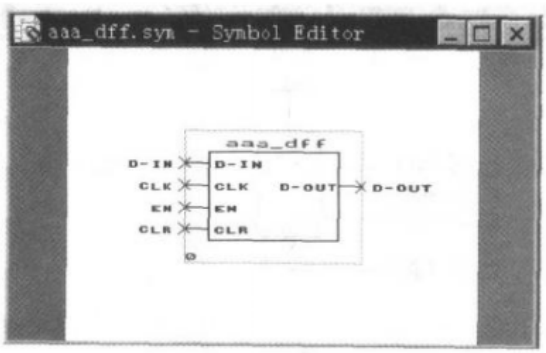


图 8.2.8 符号编辑一

通过以上几个步骤，已经成功地完成了电路原理图的输入，创建了图形输入文件(*.gdf)。

3. 文本输入

MAX+PLUS II 的文本编辑器适合于用硬件描述语言(VHDL,Verilog HDL,AHDL)编写的设计输入文件。系统对输入文件编译时会自动对语言表达的逻辑进行检查、综合,并将其映射到指定的器件中去。关于硬件描述语言 AHDL 的语法规则请参看第九章。

下面用 AHDL 语言实现一个 D 触发器(dff)的功能,具有时钟输入端、异步复位端。

```
SUBDESIGN bbb_dff
  (clk     ;input;
   reset   ;input;
   d       ;input;
   q       ;output;
  )
VARIABLE
   ss: MACHINE WITH STATES(S0,S1);
BEGIN
   ss.clk=clk;
   ss.reset=reset;
    CASE ss IS
         WHEN s0=>
              q=GND;
            IF d THEN ss=s1;
             END IF;
         WHEN s1=>
              q=VCC;
            IF ! d THEN ss=s0;
              END IF;
    END CASE;
END;
```

用硬件描述语言进行设计输入同图形输入方式基本相同,分为以下几步。

(1) 选择文本输入方式

在图 8.2.4 所示中选择 Text Editor file(文本编辑文件),单击"OK"按钮。

(2) 输入文本文件

将以上用 AHDL 编写的文本文件输入到文本编辑器窗口中。如选中菜单项 Options\Syntax Coloring,则系统将自动改变输入文本的颜色(保留字为蓝色,注释为绿色),使输入文件一目了然,减少输入错误。

(3) 保存文件并检查错误

选中菜单 File\Project\Project Save &Check,系统会自动弹出编译器窗口,如图 8.2.7 所示。这一步主要是查找输入文本文件中的语法错误,并且生成网表文件(*.cnf)。如有语法错误,则系统会弹出信息处理窗口,根据信息修改输入文件后,再重复这一步骤。

(4) 创建功能模块符号

选中菜单项 File\Create Default Symbol,系统自动对当前编辑器中的内容生成一个默认符号。可以通过选中菜单项 File\Edit Symbol 查看系统生成的默认符号,如图 8.2.9 所示。

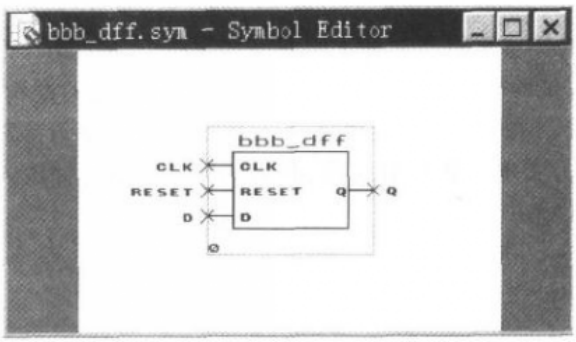


图 8.2.9 符号编辑二

4. 波形输入

MAX+PLUS II 的波形编辑器用于建立和编辑波形输入文件，还可以用来输入仿真向量和功能测试向量。波形编辑器还具有逻辑分析仪的功能，设计者可以查看仿真结果。波形输入法适合时序和重复的函数。波形输入方法如下。

(1) 选择波形输入方式

在图 8.2.4 所示中选择 Waveform Editor file(波形编辑文件)，并且在右边的下拉列表框中选择. wdf(而非. scf)，单击“OK”按钮。

(2) 编辑波形输入文件

① 创建输入/输出引脚。在波形编辑器窗口中的引脚名字(Name)、类型(Type)及赋值(Value)区域内双击，或者选中菜单项 Node\Insert Node，弹出窗口如图 8.2.10 所示。

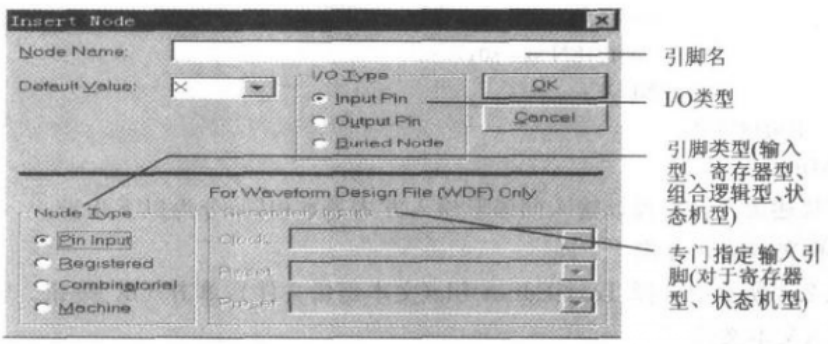


图 8.2.10 创建引脚窗口

② 编辑输入/输出波形。对输入/输出引脚进行编辑即对输入/输出波形赋值，如图 8.2.11 所示。

(3) 保存波形文件并创建默认符号

保存文件并检查错误及创建默认符号与前面所介绍的方法相同。

5. 符号编辑

符号编辑是从上至下(From top to down)设计方法的一个手段，掌握符号编辑有助于设计者从整体概念出发，把一个系统划分为各个功能模块进行设计。

以上所介绍的三种设计输入方式(图形、文本和波形)完成后，系统都可以自动生成功能模

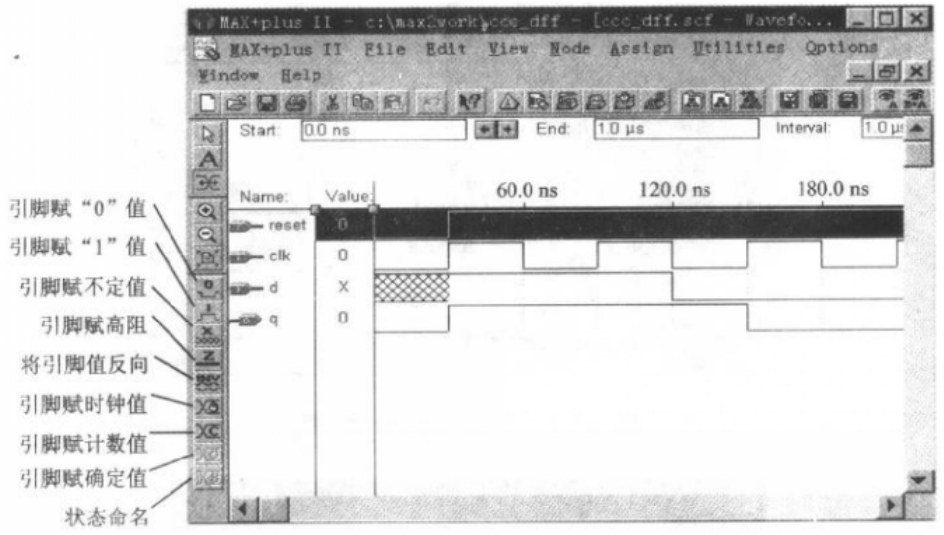


图 8.2.11　编辑波形输入文件

块符号，该符号可代替原输入文件。符号编辑正是此过程的逆过程。其步骤如下。

(1) 选择符号编辑

在图 8.2.4 所示中选中 Symbol Editor file(符号编辑文件)，单击“OK”按钮，弹出无名称的符号编辑窗口，如图 8.2.12 所示。窗口中已画出符号边框，用鼠标左键按住小圆圈移动(不能移到框外)，再用鼠标移动边框，可调整整个符号的大小。

图 8.2.12　符号编辑窗

(2) 设置输入、输出引脚

双击符号边框处，出现 Enter Pinstub 窗口，如图 8.2.13 所示。在 I/O Type 框中，选中 Input Pin，然后在 Full Pinstub Name 栏中填入输入引脚名(如 a)，单击“OK”按钮，即设置好第一个输入引脚，用同样方法可设置多个输入引脚。设置输出引脚的方法与设置输入引脚方法相同。图 8.2.14 为 3 线-8 线译码器(输入 a,b,c，输出 y0,y1,y2,y3,y4,y5,y6,y7)的符号图。

(3) 存　盘

选中菜单项 File\Save as，填入符号名(如 bcd3 - 8. sym)，单击“OK”按钮。

下面所要做的工作是向符号内添加逻辑，其方法是在前面介绍的三种输入方式中任选一种进行逻辑设计，其文件名应与符号名相同。

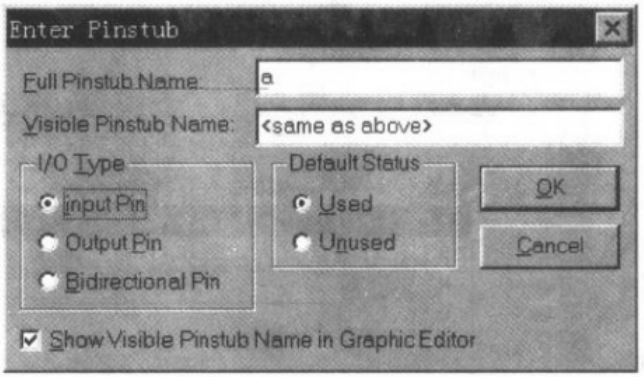


图 8.2.13 Enter Pinstub 对话框

图 8.2.14 3 线-8 线译码器

6. 底层编辑

设计完输入文件后，在编译之前还需要指定下载器件，对器件进行逻辑布局和锁定器件 I/O 端口引脚。

(1) 指定器件类型

选中菜单项 Assign\Device...，弹出对话框如图 8.2.15 所示。在 Device Family(器件类型)下拉列表框内选择器件类型(例如 MAX7000S，FILE10K)，然后在 Device 下拉列表框内指定器件(例如 EPM7128SLC84 - 6，EPF10K10LC84 - 3)。如果选 AUTO，则系统自动指定与工程文件相匹配的器件，最后单击“OK”按钮。

(2) 引脚锁定

在前面设计输入编辑窗口中只对输入、输出引脚变量名进行了定义，还没有对输入、输出引脚在指定器件中分配具体的 I/O 端口号码，对引脚分配 I/O 端口号码称之为引脚锁定。其步骤如下：

① 选中菜单项 Assign\Pin/Location/Chip，出现 Pin/Location/Chip 对话框窗口，如图 8.2.16 所示。

② 在此对话窗口中的 Node Name 栏内填入引脚名，在 Chip Resource 窗口中选中 Pin，并且填入指定器件的 I/O 端口号码；再在 Pintype 栏内选中 Input 或 Output；然后在窗口(图 8.2.16)右下边单击“Add”按钮。重复以上操作，可以为设计输入文件中的所有输入、输出引脚名分配 I/O 端口号码。最后单击“OK”按钮，这样就完成了引脚锁定。

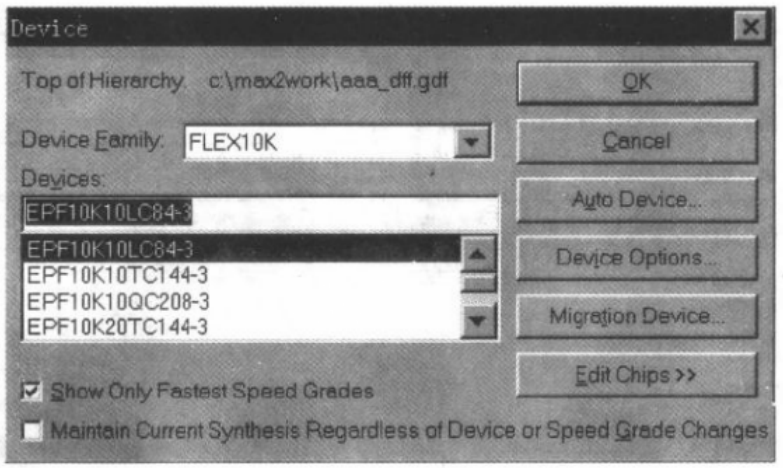


图 8.2.15　指定器件对话框

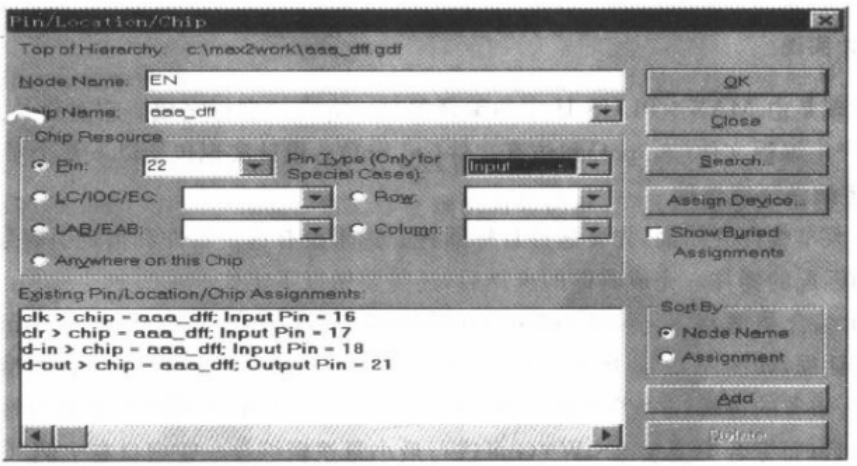


图 8.2.16　引脚锁定对话框

(3) 在底层编辑器中观察配置和加载结果

MAX+PLUS II 的 Floodplan Editor(底层编辑器)可以用来观察实际器件的内部结构，并可指定和改变器件引脚分布。在编译前后都可通过底层编辑器观察器件内部结构和器件引脚分布。并且可用底层编辑器分配引脚和配置逻辑单元。其操作步骤如下：

① 从 MAX+PLUS II 菜单中，选择 Floorplan Editor(底层编辑器)菜单项，底层编辑器将被打开，并且指定的器件结构同时显示出来，如图 8.2.17 所示。

② 底层编辑器有两种显示方式：器件视图(Device View)和逻辑阵列块图(LAB View)，可通过选择菜单项 Layout\Device View 或 LAB View 来实现。

③ 对引脚和逻辑单元可重新配置：选中菜单项 Layout\Current Assignments Floorplan，在底层编辑器的 Unassigned Nodes 框内列出了需要分配的节点和引脚名字，用鼠标把这些节点或引脚拖拽到器件视图或逻辑阵列块图指定的 I/O 引脚、节点或逻辑单元上，这样就把引脚或逻辑单元进行了重新配置。若想把新的引脚设置加载到芯片中，必须重新编译。

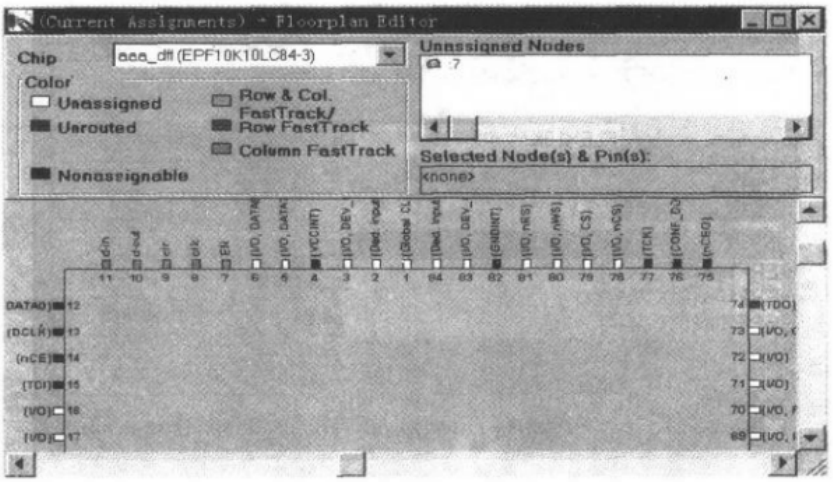


图 8.2.17 底层编辑器窗口

8.2.3 设计实现

设计实现是由 MAX+PLUS II 中的编译器(Compiler)进行完成。它对设计文件进行逻辑综合,并产生编程文件(*.pof)和仿真文件(*.snf)。设计实现包括以下几个步骤。

1. 设定编译环境参数

在打开菜单编译开始前,先要选择目标器件。如果没有选定器件和指定引脚号码,则系统自动指定相匹配的器件。完成器件的选择后,还需要做以下工作。

① 逻辑综合类型及优化设置:选中菜单项 Assign\Global Project Logic Synthesis,弹出逻辑综合对话框,如图 8.2.18 所示。在 Global Project Synthesis Style(综合类型)下拉列表中选择类型,缺省的逻辑综合类型是 NORMAL;综合类型 FAST 可以改善项目性能,但项目配置比较困难;而综合类型 WYSIWYG 可进行最小量逻辑综合。在窗口右上侧"Optimize"中将"滑块"放在适当位置:越靠左(Area),综合时芯片资源利用率越高;越靠右(Speed)则是对运行速度进行优化,但以耗用芯片资源为代价。

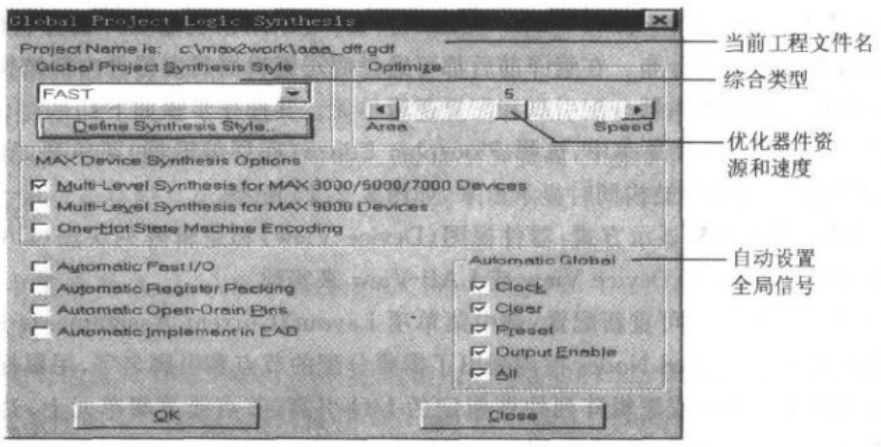


图 8.2.18 逻辑综合对话框

以下设置需要在打开菜单项 MAX+PLUS II\Compiler，弹出编译器窗口之后进行。

② 选择灵活编译命令：打开菜单项 Processing\Smart Recompile。再次编译时此命令可使系统忽略前次编译中不变的部分，而快速重新编译工程文件。

③ 打开设计医生工具：打开菜单项 Processing\Design Doctor 的开关，此时编译器窗口内增加了一个医生图标。该命令可使系统根据一定规则检验电路设计中的不确定因素。选菜单项 Processing\Design Doctor Settings，在弹出的对话框中，对于 MAX 器件选择 EPLD Rules；对于 FLEX 器件选择 FLEX Rules。

④ 打开 SNF 分析器：选中菜单项 Processing\Timing SNF Extractor，此选项将在编译过程中为后面的延时分析生成各种数据文件。

2. 编译工程文件

选中菜单项 MAX+PLUS II\Compiler 或者工具栏中的按钮，弹出编译器窗口，如图 8.2.19 所示。设定好编译环境后，可单击“Start”按钮。系统自动按照编译器窗口中的要求，从左到右依次进行各种分类编译，生成所需要的各种文件。系统自动编译分为以下七个过程：

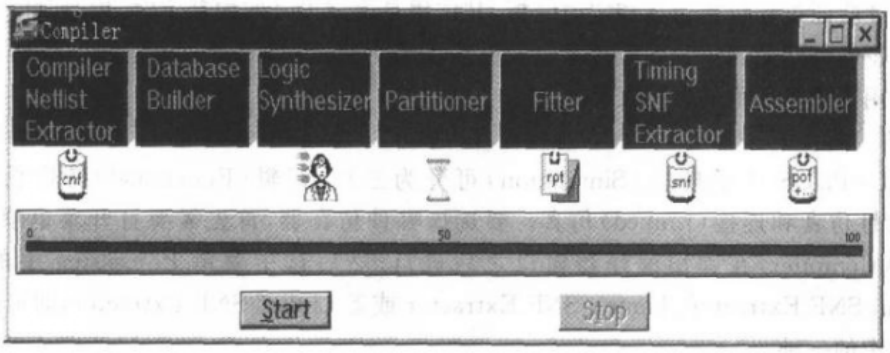


图 8.2.19　编译器窗口

① 网表(Netlist)分析器将所有设计文件转化为二进制网表文件(*.cnf)，生成层次文件；

② 数据库生成器(Database Builder)建立，用以描述整个设计的数据库；

③ 逻辑综合器(Logic Synthesizer)对整个设计进行逻辑综合，计算所有布尔等式，并优化触发器设计等；

④ 分割器选择适合当前工程设计的相应器件；

⑤ 分配实现器将逻辑设计在特定器件内实现，生成报告文件(*.rpf)；

⑥ 仿真文件生成器生成延时、逻辑或连接仿真所需要的文件(snf)；

⑦ 汇编器(Assembler)生成用以器件编程的多个目标文件。

在编译时，编译器窗口(图 8.2.19)中的状态条和进程光标表示编译过程的进行情况。由于编译是在后台进行的，因此编译过程中设计者可做其他工作。编译结束后，出现编译是否成功和编译信息窗口。如果源输入文件或编译过程有错误，则在信息窗口会显示出错信息，设计者需要对错误进行修改，然后再次编译，直到通过。

一个设计项目(工程文件)经过编译处理后，就会产生一个或多个目标文件(.pof)、SRAM 目标文件(.sof)和 JEDEC 文件(.jed)，MAX+PLUS II 的编程器使用这些文件即可对器件进

行编程。除此之外，编译器还可产生其他格式的编程文件，例如十六进制文件(.hex)、Tabular文本文件(.ttf)、串行Bit流文件(.sbf)，使用这些文件可对FIEX10K，FIEX8000等器件进行在系统编程。使用目标文件(.pof)和串行向量格式文件(.svf)可对MAX9000和MAX7000S系列芯片进行在系统编程。

3. 查看报告文件和层次显示图

双击编译器窗口中的rpt图标，显示出报告文件内容，从中可获得器件资源使用情况、器件引脚分配图和编译时间等信息。

选中菜单项MAX+PLUS II\Hierarchy Display，将出现一个窗口，窗口中显示了一个设计项目包含的所有文件以及它们之间的层次结构关系，可清晰地看到整个工程文件的结构，并且能够快捷地打开各设计文件。

8.2.4 设计验证

编译成功的设计并不一定完全正确，只有通过设计验证才能检查电路是否真正达到设计要求。设计验证过程包括仿真和定时分析，其作用是测试设计的内部定时，检验时序关系。可检查组合逻辑电路的竞争和冒险现象以及时序逻辑电路的时序、延时等指标。仿真和定时分析需要利用编译器产生的数据文件工作。

1. 仿　真

MAX+PLUS II的仿真(Simulation)可分为三种：逻辑(Functional)特性仿真、延时(Time)特性仿真和连接(Linked)仿真。要选择哪种仿真器，首先需要打开菜单项MAX+PLUS II\Compiler，在弹出编译器窗口之后进行；然后选中菜单Processing中的菜单项Functional SNF Extractor、Timing SNF Extractor或者Linked SNF Extractor，即可选中这三种仿真器中的一种。

(1) 延时仿真

它是在设计项目被综合和优化之后，对其进行仿真测试。其结果是在MAX+PLUS II的波形编辑器中显示所有节点的波形，并可观察组合逻辑电路的竞争和冒险现象，以及电路的时序、延时等指标。其步骤如下：

① 设置仿真的输入信号波形文件(.scf)。打开波形编辑器(选中MAX+PLUS II\Waveform Editor菜单项)，在波形编辑器中单击鼠标右键，在弹出的菜单中选Enter Nodes from SNF...，或选主窗口中的菜单项Node\Enter Nodes from SNF...，出现一个对话框如图8.2.20所示。可以直接在Node/Group内填入引脚名，也可单击“List”按钮，在可用引脚框内列出所有已知的引脚，通过“=>”按钮，把选中的引脚选至被选中引脚框内，单击“OK”按钮，弹出波形编辑器窗口，进行输入信号的编辑(只画出输入信号的波形)，然后把该波形文件(*.scf)存盘。

② 开始仿真。首先对工程文件进行编译，在编译器窗口中双击图标按钮，或者选中菜单项MAX+PLUS II\Simulator，弹出仿真器窗口。单击“Start”按钮，系统开始仿真运行。与编译器一样，仿真器也是在后台工作。再单击“Open SCF”按钮，进入波形编辑器窗口，通过该窗口可观察输入、输出波形以及时序、延时等。还可以在该窗口重新编辑输入波形，再存盘和编译，重复以上操作即可。

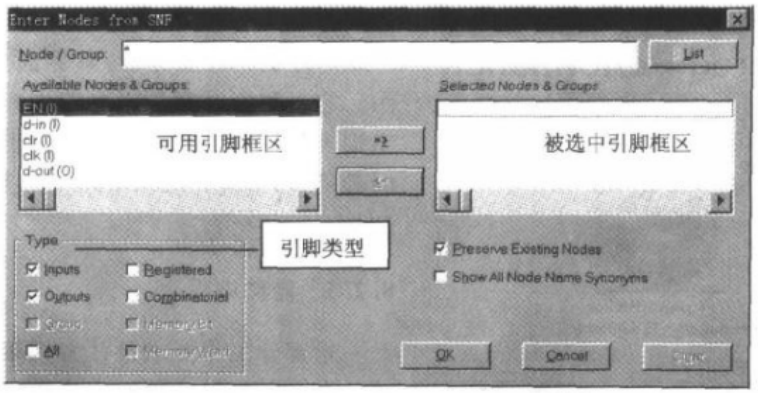


图 8.2.20　Enter Nodes from SNF 对话框

(2) 逻辑仿真(功能仿真)

逻辑仿真是在设计方案进行综合之前,测试设计输入文件的逻辑功能,可迅速知道逻辑上的错误并对其改正。MAX+PLUS II 的波形编辑器可显示逻辑功能仿真的结果,并可显示设计中所有节点的波形。其操作步骤与延时仿真相同。

(3) 连接仿真

MAX+PLUS II 可以把多个 Altera 器件的延时和逻辑仿真组合起来。这样,设计者可以对几个器件组成一起进行仿真。

2. 定时分析器

选择菜单项 MAX+PLUS II\Timing Analyzer,或单击工具栏按钮,出现定时分析器三种分析方式中的一种分析界面窗口。通过选中菜单 Analyzer 中的三个菜单项,或单击工具栏中三个按钮中的一个,就可以选择其中的一种分析方式,即(Delay Matrix)、建立/保持时间矩阵分析(Setup/Hold Matrix)或者时序电路性能分析(Registered Performance)。

延时矩阵分析:它可以计算引脚或节点之间的传输延时,只需要在设计中对引脚或节点指定起点和终点端(选菜单 Node)即可确定最短与最长传输延时。其界面如图 8.2.21 所示。

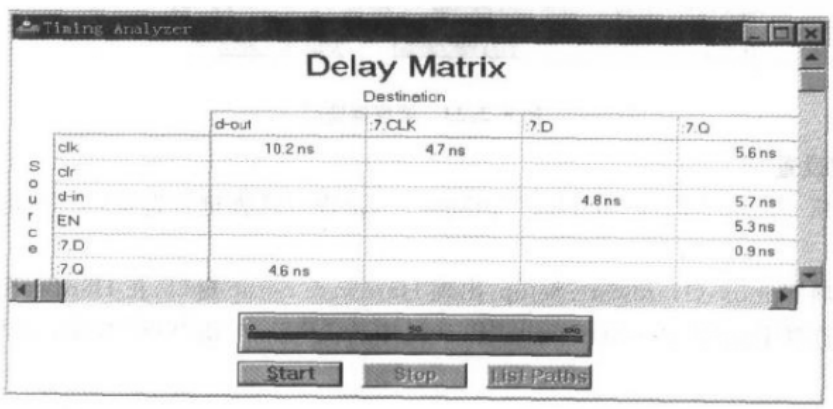


图 8.2.21　延时矩阵分析

建立/保持时间矩阵分析:它可计算在时钟作用下从输入引脚到触发器、锁存器和异步 RAM 的信号输入所需要的最少建立和保持时间以及器件引脚上信号的建立和保持时间。其

图 8.2.22 时序电路性能分析

界面与图 8.2.21 相似。

时序电路性能分析：其界面如图 8.2.22 所示。它可测算时序逻辑电路中一个触发器的输出端到另一触发器输入端的延时性能，确定最小时钟周期，从而计算出电路正常工作的最高时钟频率。

8.2.5 器件编程

这部分工作要求设计者用编译后的目标文件，下载到 Altera 器件芯片中，制成 ASIC。首先应安装好 ByteBlaster 下载电览线与计算机并行接口和目标实验板（含有 Altera 芯片），然后进行器件编程。

1. 打开编程器

选中菜单项 MAX+PLUS II\Programmer，或者单击编译器窗口中产生的图标  按钮，系统弹出编程器窗口，如图 8.2.23 所示。编程器可以用来对器件进行校验（Verify）、检测（Examine）、空白检查（BlanK - Check）以及进行功能测试（Test）等。

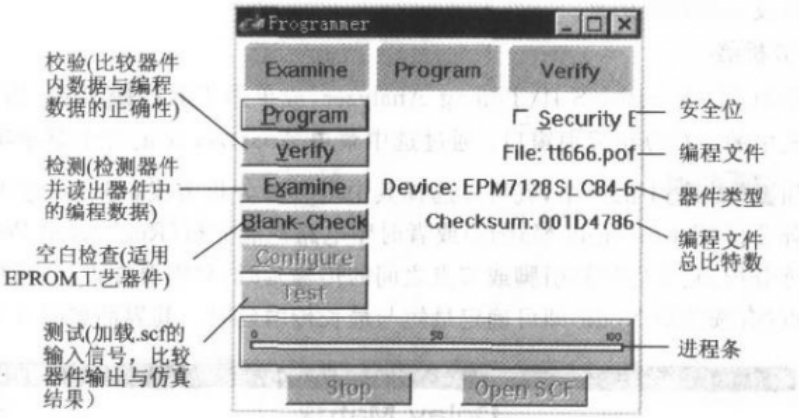


图 8.2.23 编程器窗口

2. 编程设置

选菜单项 Options\Programming Options...，出现对话框进行选择（空白检查、校验、检测、测试）。

选菜单项 Options\Hardware Setup，出现 Hardware Setup 窗口，在 Hardware Type 栏的下拉列表中选择 ByteBlaster，在 Parallel Port 栏中选 LPT1，单击“OK”按钮。此时编程设置完毕。

3. 编 程

器件编程过程是由系统自动完成。若编程器件是 Altera 公司的 MAX7000 系列或 MAX9000 系列，则编程器窗口图 8.2.23 中的 Program 按钮有效；若编程器件是 Altera 公司的 FLEX8000 系列或 FLEX10K 系列，则编程器窗口的 Configure 按钮有效。单击“Program”

或“Configure”按钮，则对器件进行编程。

8.3　ISP Synario 系统

Lattice 公司的 ISP 器件受众多开发工具软件的支持，一般这些开发工具软件集第三方著名厂家的 EDA 软件和 Lattice 公司的工具软件为一体，组成易使用、高性能的开发工具软件，ISP Synario 系统为其中之一。

8.3.1　ISP Synario 软件的特点及安装

ISP Synario 系统是一个套装软件，包括 Data I/O 的 Synario 软件和 Lattice 公司的PDS+ Fitter 适配器软件。Synario 是美国 Data I/O 公司出品的一种运行于 PC 机 Windows 环境的优秀通用电子设计工具软件，继承和发扬了 ABEL 的特点。它支持 Lattice 公司的 ISP 系列的器件，具有设计输入和仿真的功能。设计实现工具由 PDS+ Fitter 适配器软件提供，设计者可以通过设计属性适配控制参数对编译过程进行控制。

1. ISP Synario 软件的特点

ISP Synario 软件与其他开发工具软件一样，包括从设计输入、设计实现、设计仿真到器件编程所需要的可执行文件和库文件，提供了设计输入、设计实现、设计仿真及器件编程的整个过程。

① 设计输入：ISP Synario 软件的设计输入、设计实现和设计仿真都是在 Project Navigator 项目引导器集成环境下完成，该开发软件把整个设计视为一个“项目或工程”(Project)，把输入文件称为“源”(Source)。一个“项目”包含多个“源”，只有一个输入文件(源)作为顶层文件，其他输入文件作为与顶层文件有关的底层文件。Synario 软件有两个基本的输入方式：即原理图输入和文本输入(ABEL－HDL 硬件描述语言)。专业版的 Synario 软件还支持 VHDL 和 Verilog HDL 硬件描述语言。

② 设计实现：主要包括设计检验、布局、布线以及生成目标文件即 JEDEC 文件(熔丝图)。设计检验是在编译时完成的，开发软件将编译的结果报告给设计者，以便设计者修改设计。布局布线工作是由软件系统自动完成的，能以最优的方式对逻辑元件、输入/输出单元布局，并准确地实现元件之间的互联。设计实现的最后一步就是要指定编程的器件，并生成目标文件(熔丝图)。每个输入文件都可以单独进行检查、优化和编译等过程。

③ 设计仿真：包括逻辑仿真和定时仿真。在仿真前需要设计者预先建立好测试向量及测试序列，可通过报告文件形式或波形观察器检查仿真结果。

④ 下载编程：它是将开发软件生成的 JEDEC 文件装入(Download 下载)到指定的器件中。可以使用 ispCODE 和 ISP 菊花链下载软件 IDCD，通过编程电缆下载。

2. ISP Synario 的安装

① 环境要求：运行 ISP Synario 软件的要求较低，适合教学使用。

- 386/486 以上 IBM 兼容的 PC 机；
- 80 MB RAM，20 MB 硬盘空间；
- 一个 3.5 英寸的软驱，一个光驱；
- 一个并行打印口；

● Windows 3.1 以上的操作系统。

② 软件安装：首先将含有 ISP Synario 的光盘放入光驱中，找到文件 Setup.exe，并双击该文件，进入程序的安装过程。根据程序安装界面的要求进行即可。安装完成后，双击 ISP Synario 软件的图标或文件名(Synplsi.exe)，即可进入 ISP Synario 软件的主界面(ISP Synario Project Navigator)，如图 8.3.1 所示。

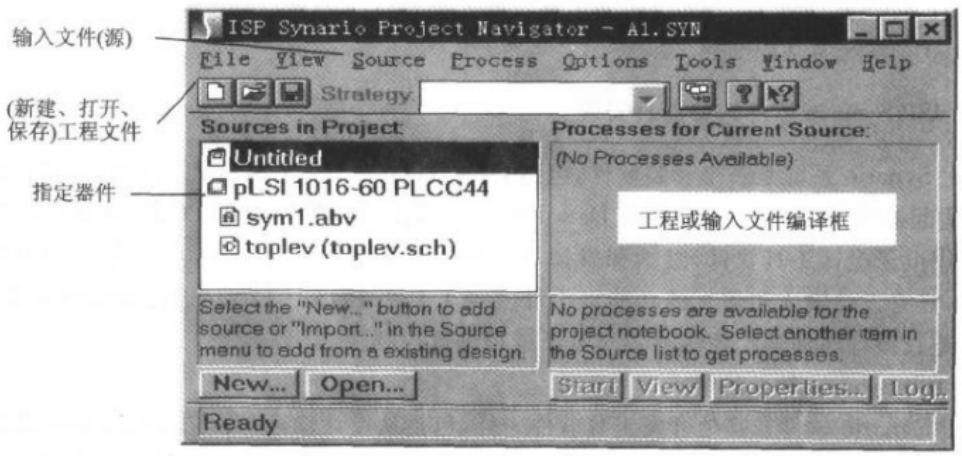


图 8.3.1 ISP Synario 的主界面

8.3.2 建立工程文件和选择器件

1. 启动 ISP Synario 软件

启动后，进入主界面，如图 8.3.1 所示。该界面分为左右两个窗口：左边窗口(Sources in Project)将显示工程的源文件；右边窗口(Processes for Current Source)将显示工程和源文件的处理过程(包括检查、优化和编译)。如果第一次使用该软件，因还未建立工程文件和源文件，这两窗口分别显示 No Project Open 和 NO Process，否则将显示前一次打开的文件名，并装入系统中。

2. 建立工程文件(.syn)

打开菜单项 File\New Project，或单击工具栏中图标(新建工程文件)按钮，出现对话框 Create New Project。在文件名栏内，填入工程文件名(*.syn)，然后单击“OK”按钮。这时主界面的顶端一行将显示工程文件名。

3. 选择器件

在主界面左边窗口中，双击 Virtual Device 项，出现对话框(Choose Device)，如图 8.3.2 所示。在 Device Kit 窗口中选择 ISP Synario Start Device 项，则在下方窗口(Device)中出现器件目录，按动滚动条选中所要求的器件，然后单击“OK”按钮即可。

4. 存储信息

选中菜单项 File\Save 或 Save as，或者单击工具栏图标(存盘)按钮，即可保存工程文件名和选择的器件。

一个工程项目由一个或多个源文件组成。这些源文件可以是原理图文件(*.sch)、测试向量文件(*.abv)、ABEL_HDL 文件或者是其他文本文件(*.doc，*.wri，*.txt)。其中有

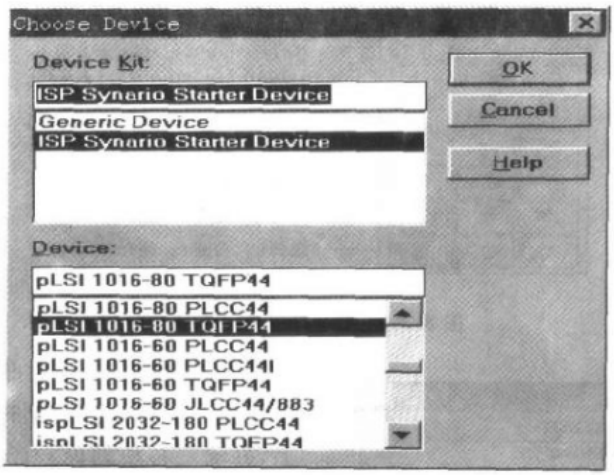


图 8.3.2　Choose Device 对话框

一个源文件可作为顶层文件。

8.3.3　原理图输入方式

1. 输入方式选择

选中菜单项 Source\New，出现 New Source 对话框，在框中列出了源文件的几种输入方式，如图 8.3.3 所示。选择 Schematic(原理图)，单击“OK”按钮，弹出原理图编辑器窗口和一个原理图文件名对话框，在对话框中填入原理图文件名(*.sch)即可。

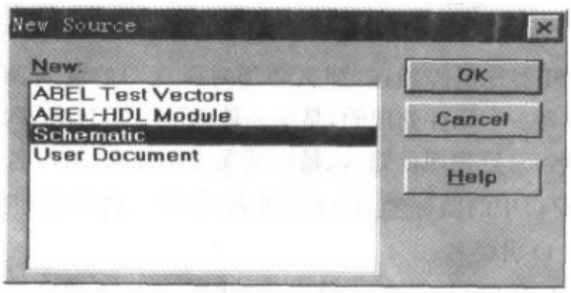


图 8.3.3　New Souce 对话框

2. 绘制原理图

从元件库中选出所需要的元件符号，将它们移到原理图编辑器窗口中的图纸上，然后用连线将它们按照设计要求连接起来。首先介绍绘图工具图标。

选中原理图编辑器窗口中的菜单项 View\Drawing Toolbar，弹出绘图工具(Drawing Toolbar)窗口，其中包括了许多工具图标，其含义如图 8.3.4 所示。它们与菜单 Add 中的许多菜单项的作用完全相同。因此可使用菜单 Add 中的菜单项来完成电路原理图的输入。

① 选择元件：打开原理图编辑器窗口中的菜单项 Add\Symbol，或单击绘图工具窗口中的图标(元件库)按钮，出现元件库窗口，如图 8.3.5 所示。其中包括算术运算电路、门电路、输入/输出电路、多路选择器和触发器等元件。

图 8.3.4 Drawing Toolbar 窗口

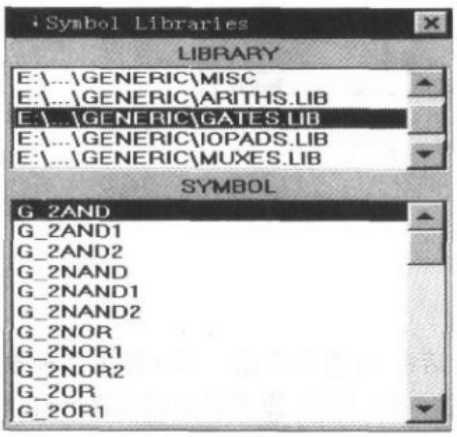


图 8.3.5 元件库窗口

② 移动元件：在元件库窗口中将鼠标点到所选择的元件，之后元件的符号就粘附在光标上，并随之移动到原理图编辑器窗口中的图纸上。单击鼠标，该元件符号就“画”在图纸上。如要进一步调整位置，可利用绘图工具窗口中的移动、旋转和翻转图标进行。

③ 连接元件：单击绘图工具窗口中的图标（连线）按钮，将光标放在所需连接的一个点上，单击鼠标左键，则该点就与连接线连上，然后将光标移至另一个连接点上，双击鼠标左键，这时两点间的连接就告完成。重复以上过程，可把所要求的元件全部画好。

注意：如原理图为顶层文件，该原理图的输入、输出端需与 ispLSI 器件引脚相连，那么必须对原理图的输入、输出端连接 I/O 缓冲电路。I/O 缓冲电路是作为一个元件处理的，是在元件库的子库 IOPADS 中。

④ 定义 I/O 引线名：对原理图的输入、输出端需定义输入、输出名。其步骤如下：

● 单击绘图工具窗口中的图标（I/O 引线名）按钮。此时原理图编辑器窗口下方的状态栏给出设计者输入 I/O 引线名。

● 填入 I/O 引线名，按“Enter”键，引线名就粘附在光标上随之移动。移至需要命名的 I/O 端（红框上），单击鼠标左键，引线名就被安放在那里。也可用连线与 I/O 端相连。

● 重复以上步骤，可把全部输入、输出端点定义为 I/O 引线名。

⑤ 定义 I/O 标记：在对输入、输出端点定义 I/O 引线名的同时，还应该定义引线名的性质（输入、输出或双向）。

● 单击绘图工具窗口中的图标（I/O 标记）按钮，弹出 I/O Markers 选择框，根据 I/O 引线属性选中 Input、Output 或 Bidir。

● 再将光标移至需定义的引线名，单击鼠标左键，就会出现一个 I/O 标记，标记框内是引线名。这样可把所有引线名定义对应的 I/O 标记。

至此电路原理图的输入基本完成。图 8.3.6 所示为一个举例。最后保存原理图输入文件（选中菜单 File\Save），并可退出原理图编辑器。

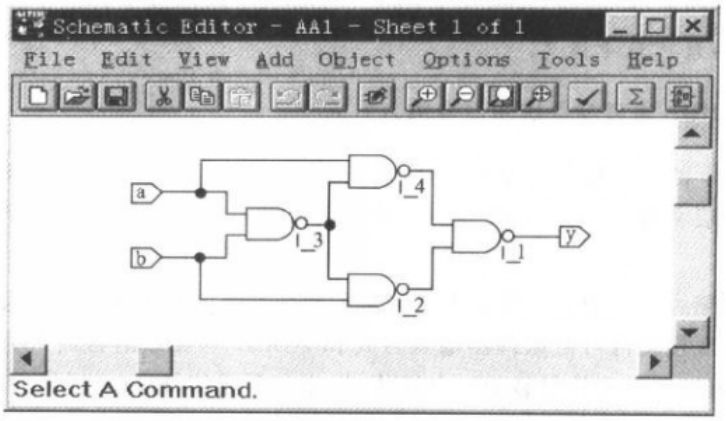


图 8.3.6　原理图举例(aa1.sch)

3. 编译原理图

电路原理图输入完成后，回到主界面，选中原理图源文件名。双击处理过程窗口中的 Compile Schematic 项，系统对该源文件进行编译。如编译通过，则在该 Compile Schematic 项前出现一个绿色的记号“√”；如出现一个黄色的记号“!”，则表示有警告问题；如出现一个红色的记号“×”，则表示源文件编辑有错误，编译没通过，并将显示出错误所在及类型。

在编译的同时也进行了逻辑简化，可选择 Reduced Equations(化简方程)，将编译结果以及逻辑方程的形式表现出来。

4. 建立功能模块

如果原理图源文件作为底层文件，那么可把原理图文件快速封装成为一个功能模块，用一个逻辑符号表示，作为可供反复调用的功能元件，以便放置在更高一层的原理图中。其建立模块符号的方法为：在电路原理图输入完成后，选中菜单项 File\Matching Symbol，此时逻辑符号已经建立，并且已存入元件库的子库 Local(本地元件库)中。可作为一个元件反复调用。

8.3.4　ABEL - HDL 语言输入方式

该输入方式是在已经建立了工程项目和已经选定器件的基础上进行的。关于 ABEL - HDL 语言语法和规则详见第九章。

1. 输入方式选择

选中菜单项 Source\New，出现 New Source 对话框。在框中选择 ABEL - HDL Module，单击“OK”按钮，弹出文本编辑器窗口和一个 New ABEL - HDL Source 对话框，在对话框中填入模块名称和文件名(*.abl)(与模块名称相同)。单击“OK”按钮，进入文本编辑器窗口。

2. 编辑 ABEL 源文件

在文本编辑器窗口中已出现 ABEL - HDL 设计框架和上述键入的模块名。根据设计要求和 ABEL - HDL 语言规则，写入描述电路逻辑关系的文本语句。完成文本编辑后，保存该源文件，进入主界面(工程项目管理器)。

3. 编译 ABEL 源文件

在主界面的 Sources in Project 窗口中选中编译的 ABEL 源文件。在右边处理窗口中双击 Compile Logic 项，源文件就被编译。如编译通过，则在 Compile Logic 项前边出现绿色

“√”标记；如编译没通过，将在 Synario Report Viewer 中指出源文件的错误，并显示出来，通过修改重新编译，直到编译通过。

8.3.5 ABEL－HDL 语言与原理图混合输入方式

采用自顶向下的方法进行系统设计时，首先对整个系统进行方案设计和功能划分，设计出模块化结构的顶层文件，每个模块代表一个功能块或者子系统。然后对每个功能模块进行底层文件设计。

由于原理图输入方式有利于逻辑接口关系和模块化结构的设计，顶层文件常采用原理图输入方式设计。底层文件可由原理图输入方式和 ABEL－HDL 语言输入方式设计。为了顶层文件的调用，可把底层文件封装为一个模块，作为一个元件符号（宏元件）反复使用。其操作步骤如下。

1. 建立模块符号（宏元件）

进入原理图编辑器窗口中，打开菜单项 Add\New Block Symbol，弹出一个对话框，填入模块名（Block Name）、输入端、输出端及双向端，单击“Run”按钮。系统自动生成一个元件符号，并以模块名的名称存储在元件库的 Local（本地）库中。这时建立好的模块符号还是空的，需要写入源文件（原理图输入或文本输入的底层文件）。

2. 建立顶层原理图（顶层设计）

按照绘制电路原理图的方法，从元件库中调出新建的模块符号和其他元件符号，完成电路原理图（顶层文件）的设计。保存文件，回到主界面（工程项目管理器）。在主界面的左窗口中出现新建模块符号的模块名，由于该模块现在还是空的，所以它的前边有一个红色“?”标记。

3. 编辑模块符号的源文件（ABEL 输入方式）

双击主界面左窗口中带“?”标记的模块名，出现 New Source 对话框。选择 ABEL－HDL Module，进入文本编辑器窗口，按照 ABEL－HDL 语言输入方式编辑源文件。存盘退出。

4. 编译顶层原理图

在编译顶层原理图之前，先对新建的模块符号进行编译。然后编译顶层文件。

8.3.6 功能仿真和波形显示

ISP Synario 软件同时提供逻辑功能仿真器和波形观察器。在仿真前需要设计者预先建立好测试向量及测试序列。对原理图输入文件进行仿真时，需要单独建立一个仿真的源文件即测试向量的文件；对 ABEL－HDL 输入文件仿真时，测试向量可以作为 ABEL 源文件中的一部分，也可以单独建立一个测试向量文件。

1. 建立仿真测试向量

选择菜单项 Source\New，弹出如图 8.3.3 所示的对话框。选择 Simulation Test Vectors（仿真测试向量）输入方式，并单击“OK”按钮，弹出文本编辑器窗口和一个 New File 对话框。在对话框中填入测试向量的文件名（一般与对应的原理图文件或 ABEL 文件同名），然后单击“OK”按钮，进入文本编辑器窗口编辑输入测试向量，测试向量文件的输入方式与 ABEL 语言的输入方式相同。详见第九章 ABEL－HDL 硬件描述语言。例如，图 8.3.7 为一个原理图（图 8.3.6 所示）的测试向量文件（aa1.abv）。

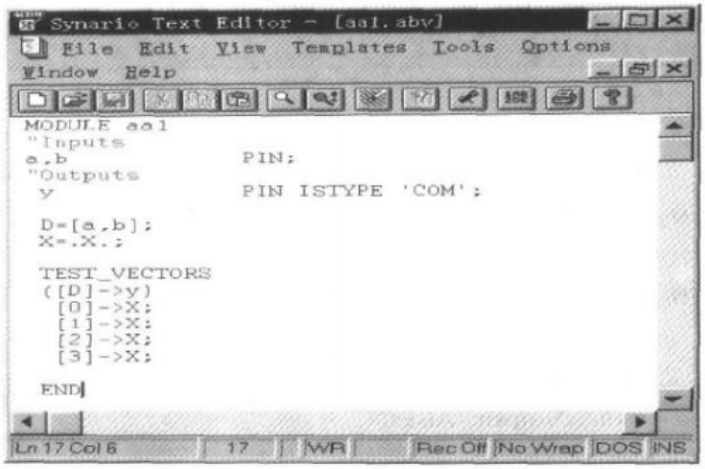


图 8.3.7　测试向量文件举例(aa1.abv)

2. 编译测试向量的源文件

在系统主界面(工程项目管理器)右边处理窗口中,双击第一项 Compile Test Vectors 进行编译。如编译通过,则在该项前出现绿色的"√"标记。

3. 逻辑仿真

双击处理窗口中的 Simulate Equations 项,系统即对设计进行逻辑仿真。仿真通过,在 Simulate Equations 项的前面出现绿色"√"标记,否则出现红色"×"标记,如图 8.3.8 所示。通过仿真报告(Equation Simulation Report),可查看仿真结果或出错情况。

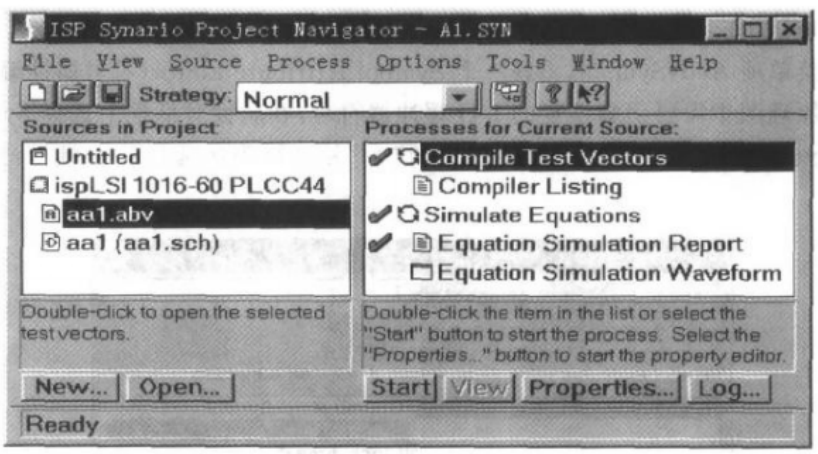


图 8.3.8　逻辑仿真

4. 波形图显示

双击处理窗口中的 Equation Simulation Waveform 项,出现波形观察器窗口,如图 8.3.9 所示。接着选中菜单项 Edit\Show,弹出一个对话框,选择需要显示波形的结点或输入/输出端,单击"Show"按钮,即可在波形观察器中显示其波形。图 8.3.9 为一个设计(aa1.sch,aa1.abv)的仿真波形图。

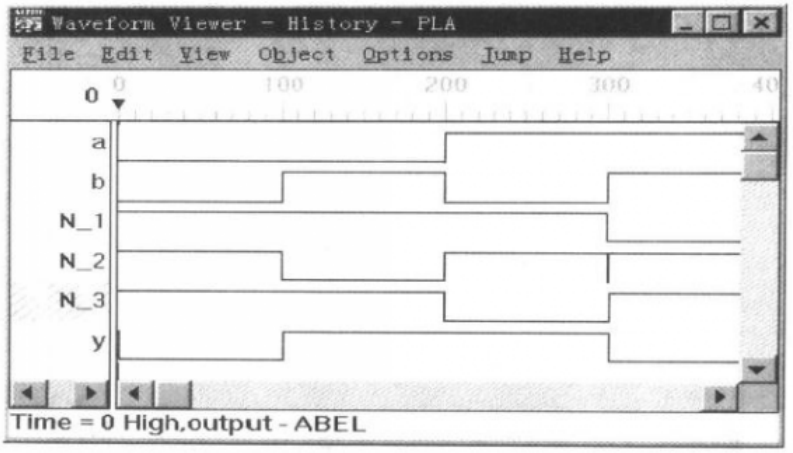


图 8.3.9 仿真波形图举例

8.3.7 引脚锁定、JED 文件生成及下载编程

在完成以上的输入源文件的设计、编译和器件指定后，还需要指定器件的引脚，生成目标文件(JED 文件)，并下载到器件芯片中，即实现在系统可编程器件的编程。

1. 引脚锁定

引脚锁定是把输入文件中的输入/输出信号与器件的引脚端口相连，而指定输入/输出信号的引脚号。对于用原理图编写的顶层文件或单个源文件，在其输入/输出信号与 I/O 标记(I/O Marker)之间，必须加入 I/O 缓冲器(I/O Pad)。引脚锁定就是在 I/O 缓冲器中指定器件的引脚号。其操作步骤如下：

① 选中菜单项 Add\Symbol，出现一个 Symbol Attribute Editor(符号属性编辑)对话框。

② 单击原理图中需要引脚锁定的 I/O 缓冲器符号，在属性对话框中选择 Synario Pin 属性，并且把 SynarioPin Attribute(属性)栏中的“*”号改写为所选择的引脚号，如图 8.3.10 所示。

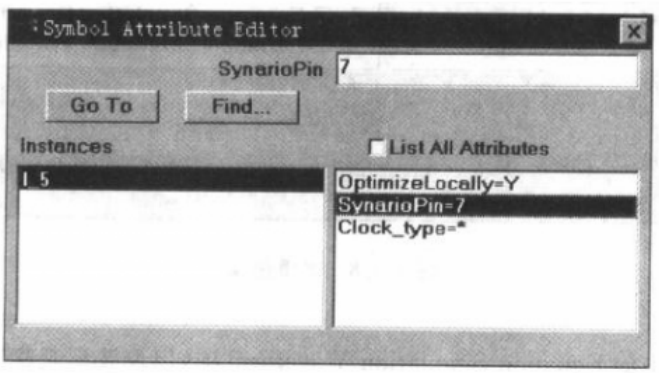


图 8.3.10 Symbol Attribute Editor 对话框

③ 关闭对话框，此时所选择的引脚号便出现在相应的 I/O 缓冲器符号内。图 8.3.11 为一个举例。其中元件名为 I_5，I_6，I_7 的是 I/O 缓冲器，填入的对应引脚号为 7，4，3 。

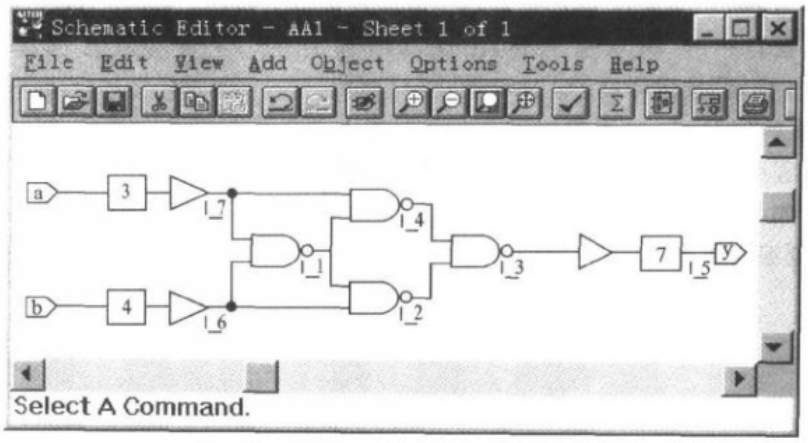


图 8.3.11　一个举例

以上锁定引脚的方法是在原理图中定义 I/O 缓冲器的属性，来达到分配输入/输出信号的引脚的目的。还有另一种锁定引脚的方法：利用编辑锁定文件(*.ppn)的方法来实现输入/输出信号的引脚分配。

2. 生成 JED 文件

在系统的主界面源文件窗口中选择已指定的器件(如 ispLSI1016 - 60 PLCC44)，此时右边的处理窗中出现一些项目，如图 8.3.12 所示。将这些项目自上而下逐一运行。最后选择 ISP Synario Fitter Report(适配报告)项，将显示布线利用率、引脚分配等信息。如果设计者事先没有锁定引脚，则系统将自动进行分配。

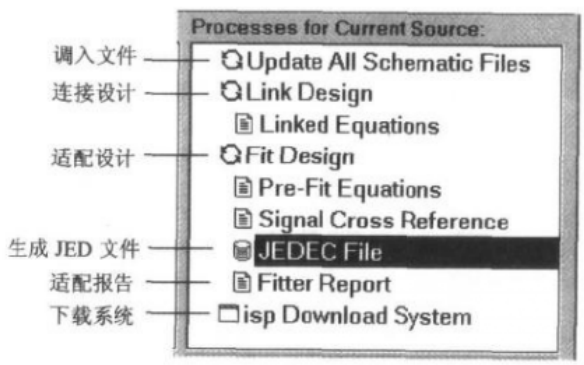


图 8.3.12　处理窗口

3. 下载编程

Lattice ISP 器件的在系统编程能够在多种平台上通过多种方式来实现。在此介绍最常用的基于 PC 机 Windows 环境的菊花链式在系统编程方法。它是通过 ISP 菊花链编程软件把 JEDEC 数据文件下载到单片或多片 ISP 器件内。其编程过程如下：

① 启动 ISP 菊花链编程软件，即双击 IDCD 图标，出现 LSC ISP Daisy Chain Download...操作窗口。

② 将编程电缆一端插入计算机的并行接口上，再把电缆另一端连接到 ISP 目标系统板(含有 ispLSI 器件)上，然后给目标系统板上加入电源。

③ 选中菜单项 Configuration\Scan Board，或者单击工具栏中的 SCAN 图标，即刻建立结构文件。双击窗口中的“Browse”按钮，出现 Browse JEDEC File 对话框，选择 JEDEC 文件（*.jed），单击“确定”按钮即可。

④ 选中菜单项 Command\Run Operation、或者单击工具栏中的小人跑步图标，开始编程。在编程过程中每个器件的 Status 栏目内会显示操作进程和结果，出现 Pass 表示编程完毕并且已通过。如果出现 Fail（失败），应该根据 Message（信息）窗口的提示，检查和修改错误，重新编程，直到编程通过。

第九章　可编程逻辑器件的硬件描述语言

9.1　硬件描述语言概述

可编程逻辑器件的广泛应用，为数字系统的设计带来极大的灵活性，改变了传统的数字系统的设计方法、设计过程以及设计观念，使硬件设计如同软件设计那样方便快捷。现代的EDA工具软件已突破了早期仅能进行PCB印刷版图的设计或功能模拟，配备了系统设计的全部工具。如提供了多种能兼用和混合使用的逻辑描述输入方式，即原理图输入、波形输入和硬件描述语言文本输入等方式。

硬件描述语言HDL(Hardware Description Language)是一种用形式化方法来描述数字电路和设计数字逻辑关系的语言，其中包括布尔方程方式、真值表和状态图描述方式。它可以使设计者利用这种语言来描述自己的逻辑设计思想，然后通过EDA工具进行仿真，自动优化综合，生成门级电路，再用ASIC或可编程逻辑器件实现其功能。目前在美国硅谷约有80 %的ASIC和可编程逻辑器件是采用HDL方法设计的。

硬件描述语言的发展至今已有20年的历史，并成功地应用于设计的各个阶段：仿真、验证、综合。到20世纪80年代时，已出现上百种硬件描述语言，它们一般各自面向特定的设计领域和层次，因而这些众多语言使设计者无所适从。随着发展，现在主要硬件描述语言有VHDL，Verilog HDL，ABEL－HDL和AHDL。

1. VHDL语言

VHDL的英文全名是Very－High－Speed Integrated Circuit HDL(超高速集成电路硬件描述语言)。它源于美国国防部提出的超高速集成电路计划，其目的是为了在各个承担国防部订货的集成电路厂商之间建立一个统一的设计数据和文档交换格式。1987年底，VHDL被IEEE(The Institute of Electrical and Electronics Engineers电气和电子工程师协会)采纳为硬件描述语言标准IEEE1076。

VHDL是一种全方位的硬件描述语言，包括从系统到电路的所有设计层次，主要用于描述数字系统的结构、行为、功能和接口。VHDL的语言形式和描述风格十分类似于一般的计算机高级语言。其特点如下。

● VHDL具有很强的行为(功能)描述能力，从而决定了它成为系统设计领域最佳的硬件描述语言。强大的行为描述能力是避开具体的器件结构，从逻辑功能上描述和设计大规模系统的重要保证。

● 具有层次结构性。VHDL的程序结构特点是将一项工程设计或称设计实体(一个元件、电路模块或一个系统)分成外部(即端口)和内部。外部描述系统输入/输出接口和有关参数；内部描述系统内部的结构和行为状态，并且可形成功能模块，以备其他设计调用。这种分开描述有助于层次化的设计。

● VHDL语言是一种并行语言，其编程思想与传统顺序执行的计算机语言(如C，Pascal)

有很大的区别。

● VHDL对设计的描述具有相对独立性，设计者可以不懂硬件的结构，也不必管最终设计实现的目标器件是什么，而进行独立的设计。

● 由于VHDL具有类属描述语句和子程序调用等功能，对于已完成的设计，在不改变源程序的条件下，只需改变类属参量或函数，就能轻易改变设计的规模和结构。

2. Verilog HDL语言

Verilog HDL是在1983年由GDA((Gate Way Design Automation)公司的Phil Moorby首创的。1986年Moorby提出了用于快速门级仿真的Verilog XL算法，促使Verilog HDL语言得到迅速发展。1989年Cadence公司收购了GDA公司，Verilog HDL成为Cadence公司的私有财产。1990年Cadence公司公开Verilog HDL语言。基于Verilog HDL优越性，IEEE于1995年制定了Verilog HDL的IEEE标准，即Verilog HDL 1364—1995。

近年来，关于VHDL语言和Verilog HDL语言在EDA界一直争论不休。这两种语言各有所长，市场占有量也相差不多。Verilog HDL是专门为ASIC设计而开发的，通常适于寄存器传输级(RTL)和门电路级的描述，是一种较低级的描述语言。而VHDL语言通常适于行为(功能)级和寄存器传输级(RTL)的描述，是一种高级描述语言，最适合于描述系统功能，但几乎不能直接控制门电路的生成。大多数EDA软件都支持这两种硬件描述语言。

3. ABEL－HDL和AHDL语言

这两种硬件描述语言是由相应的EDA开发软件所限定使用的语言，还没有成为IEEE标准。它们适合于寄存器传输级(RTL)和门电路级的描述。它们的特点和受支持的程度远远不如VHDL和Verilog语言。Verilog是从集成电路设计中发展而来，语言较为成熟；而ABEL和AHDL语言是从可编程逻辑器件的设计中发展而来，具有使用灵活、格式简洁、编译要求宽松等特点。

ABEL语言是由Data I/O公司开发的，虽然有不少EDA软件(如ispEXPERT，Synario，Foundation)支持，但提供ABEL－HDL综合器的EDA公司仅Data I/O一家。AHDL语言完全集成于Altera公司的MAX+PLUS II的软件开发系统中，只能在该开发软件中进行编译和调试。

9.2 AHDL硬件描述语言

AHDL是一种模块化的高级语言。它集成于MAX+PLUS II的软件开发系统中，特别适合于描述复杂的组合逻辑、组运算、状态机和真值表。AHDL文件作为一种文本文件，既可以用MAX+PLUS II提供的文本编辑器，也可以用其他文本编辑器来建立文本设计文件(.tdf)。但是MAX+PLUS II文本编辑器更适合进行文本编辑、编译和调试等工作，尤其是在信息处理器中对错误有自动定位的功能，使调试十分方便。MAX+PLUS II编译器还可以产生AHDL文件设计的报告文件(.tdx)和文本设计输出文件(.tdo)。

在AHDL文件中包含很多有特色的段和语句，并且包含许多用以在行为语句中对逻辑进行描述的元素。可以使用AHDL建立完整层次的工程设计项目，或者在一个层次的设计中混合使用AHDL文件和其他类型的设计文件。

9.2.1　AHDL 的基本元素

AHDL 具有计算机编程语言的一般特性，其语言元素是编程语句的基本单元。准确理解和掌握 AHDL 元素的含义和用法是十分重要的。

1. AHDL 的数值

数值被用来在逻辑表达式、真值表、状态机和等式中指定常量值。AHDL 支持十进制、二进制、八进制和十六进制数的所有组合。采用不同前缀 B(二进制)、Q 或 O(八进制)、X 或 H(十六进制数)来区分，接着用双引号把数值包括起来。

例如：以下是正确的写法：

56　　B"011010"　　B"0110X1X10"　　Q"4671223"　　H"123AECF"

以下规则也适用于 AHDL 的数值。

① MAX+PLUS II 编译器总是把逻辑表达式中的数值翻译为二进制数。

② 在表达式中的一个单一节点不能用数值赋值，必须用常量 VCC 和 GND(关键字)来赋值。VCC 表示信号的高电平和逻辑"1"；GND 表示信号的低电平和逻辑"0"。

③ 字符常量可由 Constant(常量)语句定义。

2. 符号名

符号名由一串字母数字符号组成，与字母的大小写无关，长度不得超过 32 个字符，字母不分大小写。符号名又分为带引号的和不带引号的符号名，带引号的是把符号名括引在单引号内。因此定义符号名时应按以下规定。

● 不带引号的符号名由字母 a～z、A～Z、0～9、斜线(/)、下划线(_)等符号组成，但是不能是关键字、保留标识符；还有不能只由数字(0～9)符号组成。

● 带引号的符号名由字母 a～z、A～Z、0～9、斜线(/)、半字线(-)、下划线(_)等符号组成。关键字可被括在单引号内作为符号名使用，但是单引号内不能是保留标识符。

例如，合法的不带引号和带引号的符号名：

a　　/a2　　'-bar'　　'table'　　'1221'

不合法的不带引号和带引号的符号名：

node　　-foo　　55　　'bowling4 $'　　'has a space'

在 AHDL 中有三种类型的符号名。

① 用户定义的标识符。它们在 AHDL 文件中被用来对如下部分进行命名：内部和外部节点、常量、状态机变量、状态名和状态位、实例。

② 子设计名：是用户为下层设计文件定义的名称，其长度不得超过 8 个字符。该名称必须与相应的 TDF 文件同名。

③ 端口名：是为逻辑函数的输入和输出指定的端口名称。

3. 关键字和保留标识符

在 AHDL 语句的开始、结尾及中间过程都需要使用关键字。保留标识符是 AHDL 为一些专门用途所保留的名称，用户不能随意使用这些标识符。应避免在设计文件中使用保留标识符和关键字作为节点名、常量名和端口名。

使用保留标识符和关键字的区别是：关键字被括在单引号中可当作符号名使用，而保留标识符则不能；但它们都可以在注释中任意使用。它们与字母的大小写无关，Altera 公司建议用·

大写字母来写关键字，以便阅读文件。

(1) 关键字 (Reserved Keywords)

AND	FUNCTION	OUTPUT
ASSERT	GENERATE	PARAMETERS
BEGIN	GND	REPORT
BIDIR	HELP_ID	RETURNS
BITS	IF	SEGMENTS
BURIED	INCLUDE	SEVERITY
CASE	INPUT	STATES
CLIQUE	IS	SUBDESIGN
CONNECTED_PINS	LOG2	TABLE
CONSTANT	MACHINE	THEN
DEFAULTS	MOD	TITLE
DEFINE	NAND	TO
DESIGN	NODE	TRI_STATE_NODE
DEVICE	NOR	VARIABLE
DIV	NOT	VCC
ELSE	OF	WHEN
ELSIF	OPTIONS	WITH
END	OR	XNOR
FOR	OTHERS	XOR

(2) 保留标识符(Reserved Identifiers)

CARRY CASCADE CEIL DFFE DFF EXP FLOOR GLOBAL JKFFE JKFF LATCH LCELL MCELL MEMORY OPENDRN SOFT SRFF SRFFE TFF TFFE TRI USED WIRE X

这些保留标识符包括了所有缓冲器、触发器和锁存器等基本元件的名称以及预定义的逻辑级 X。

4. 节点和组

节点可以看成是单个信号。组是多个节点的集合，被当作一个整体来操作。在逻辑表达式和等式中，常把相同类型的符号名和端口名称当作组来说明和应用。一个组最多可包括256个成员(或位)。在设计文件的逻辑段或变量段中，组可由许多节点组成，并且在表达式和等式中一个节点和常量 VCC，GND 可以被复制成一个组。

(1) 单值域组

它是由一个符号名或端口名后跟一个括在方括号中的整数域组成的。整数域可由数字或算术表达式表示，在它们中间用二个断续点(..)隔开，并且括在方括号内。例如：a[4..1]表示 a4，a3，a2，a1 这四个节点。域一般按降序排列，如果要按升序或混合排列，则必须用 Options 语句说明。

如果一个组在前面已被定义了，则可利用[ ]来表示该组，例如，a[4..1](已被定义过)也可用 a[ ]表示。为了表示一个组里的单个符号名或某个节点，可以把单个数字放在方括号内，如

a[3]；也可在符号名后紧接一个数字，如 a3 与 a[3] 的意义相同。还有其他表示组的方式，例如：

a[2＊2..2－1]，a[B"100"..B"001"]都表示一个具有 a4，a3，a2，a1 的组。

(2) 双值域组

它是由一个符号名或端口名后跟括在方括号中的两个整数域组成的，如 b[2..1][5..3] 表示具有成员 b[2][5]，b[2][4]，b[2][3]，b[1][5]，b[1][4]，b[1][3]的组。也可用如下方法来表示一个组的成员：b2_5，b2_4，b2_3，b1_5，b1_4，b1_3 。

(3) 序列组

一个序列组是由一组符号名、端口名或数字组成的。它们之间用逗号分隔，并且被括在圆括号中。例如(a，b，c)和(a，b，c[5..1])都是合法的组名。这种序列组对于指定端口名是非常有用的。例如一个 DFF(D 触发器)类型的变量 reg 的输入端口可以被写作 reg.(d，clk，clrm，prm)。序列组中的逗点还可用于保持一个未分配组成员的位置，如(a4，，a1)。

a[4..1]与(a4，a3，a2，a1)表示的意义完全相同。a[3..1]和(a4，a2，a1)表示 a[4..1]的部分成员。

(4) 给组赋值

在逻辑表达式中，一个组可以被表达式、另一个组、单个节点、常量 VCC 与 GND、数值 1 与 0 等赋值。每种情况下组的赋值是不同的。

① 如果一个组被设置等于 VCC 或 GND，那么该组内的每个成员(位)都被赋值为 VCC 或 GND。例如，a[2..0]＝VCC，则表示 a2，a1，a0 都为 VCC。

② 如果一个组被设置等于一个节点，那么该组内的所有成员都与这节点相连。如 a[2..0]＝b。

③ 如果一个组被设置等于 1(十进制数)，例如 a[2..0]＝1，在编译时，数值 1 将被扩展为 B"001"，所以只有 a0 被连到 VCC 上，其他成员为 GND。

④ 如果一个组被设置等于另一个组，例如 a[2..0]＝b[5..3]，那么表示 a2＝b5，a1＝b4，a0＝b3。如果 a[3..0]＝b[4..3]，那么表示 a3＝b4，a2＝b3，a1＝b4，a0＝b3。

5. 符　号

在 AHDL 中有各种预先定义的符号。包括一般符号、运算符号(算术和逻辑)和比较符号。它们在 AHDL 文件中有各种不同的含义和作用。

(1) AHDL 的一般符号

AHDL 的一般符号及功能如表 9.2.1 所列。

(2) 算术运算符和比较运算符

表 9.2.2 列出 AHDL 的算术运算符和比较运算符及其使用说明。它们可分为在逻辑表达式中的运算符和在算术表达式中的运算符。其形式相似，含义不完全相同。

当 LOG2 的结果不是整数时，会自动取大于该数的下一个整数，如 LOG2(258)＝9。

在逻辑表达式中的算术运算符(＋ 与－(一元)、＋、－)的用法有以下规定：

- 操作数必须是节点组或数值；
- 如果两个操作数都是节点组，那么这两个组的长度必须相同；
- 如果两个操作数都是数值，那么短的数值将带符号扩展至另一个操作数的长度；
- 如果一个操作数是数值而另一个是节点组，那么这个数值将被裁至或带符号扩展至节点组的长度。如果数值的某有效位被裁去，那么系统编译时会发出错误信息。

另外，如果当逻辑等式的右侧有两个节点组相加时，那么可以在每个组的左边加入一个0来对各组的长度进行扩展。用这种方法可以给等式左侧的组提供一个附加位，用它来代表进位信号。例如(co,s[7..0])=(0,a[7..0])+(0,b[7..0])。

比较符也分为逻辑比较符和算术比较符。逻辑比较符能够比较单独节点、节点组以及不带无关项(X)的数值。如果对节点组或数值进行比较，那么节点组必须具有相同的长度。系统编译时对组是按位进行比较。若比较结果为真，则返回VCC；否则返回GND。算术比较符只能用来对节点组和数值进行比较，而且组的长度必须相同。编译器对节点组作的是无符号值的比较，也就是说每个节点组都作为一个正二进制数与另一组进行比较。

表9.2.1 AHDL符号

符 号	功 能
_(下划线) -(半字线) /(正向斜线)	组成符号名的字符
--(双半字线)	注释的开始，直到这一行的结尾
%(百分号)	在两个%之间为注释
( )(左和右圆括号)	① 用于序列组的名称； ② 用于参数语句、子设计段和函数原型语句中； ③ 在真值表语句中选择性括引输入和输出； ④ 括引状态机中的状态位和状态； ⑤ 在表达式中最优先级的运算； ⑥ 选择性地括引判断语句中的条件
[ ](左和右方括号)	括引单值或双值域数组的数域
‘ ’(单引号)	带引号的符号名
“ ”(双引号)	括引标题语句、参数语句和判断语句中的字符串、函数语句中的文件名以及括引非十进制数字
..(省略号)	用在一个区域的MSB和LSB之间
;(分号)	用在AHDL语句和段的末尾
,(逗号)	分隔序列组和表中的各成员
:(冒号)	在说明语句中分隔符号名和类型名
=(等号)	① 为输入端口、语句中的参数设置默认值； ② 在逻辑等式中赋值； ③ 为状态机状态赋值； ④ 在选择语句中为选择指定的设置； ⑤ 把一个信号同内部直接引用中的一个端口相连起来采用给端口联合起名

表 9.2.2　算术运算符和比较运算符

运算符	例　子	说　明
+(一元)	+1	正(一元算子)
-(一元)	-1	负(求补)
!	!a	非
^	a^2(a^2)	乘方(a^2)
MOD	4MOD2	求模
DIV	4DIV2	除
*	a * 2	乘
LOG2	LOG2(4-3)	以 2 为底的对数
+	1+1	加
-	1-1	减
==(数值)	5==5	数值相等
==(串)	"a"=="b"	串相等
!=	5!=4	不等
>	5>4	大于
>=	5>=4	大于或等于
<	a<b+2	小于
<=	a<=b+2	小于或等于
?	(5<4)? 3:4	三元算子

(3) 逻辑运算符

表 9.2.3 列出逻辑运算符及其使用说明。从表中可以看出，一个逻辑运算符有两种书写方式，使用时可任选一种。逻辑运算符在使用时其含义如下：

① 如果"非"运算符后的操作数是节点组，那么该组中的每个成员求反，如!a[3..1]被解释为(!a3,!a2,!a1)；如果操作数是数值，那么对这个数值的二进制数的每位求反，如!B"1001"，即为 B"0110"。

② 如果有相同长度的两个节点组进行逻辑运算，那么其运算符将对两个节点组中相对应的两个成员进行运算，也就是对两组按位运算。例如 a[3..1]&b[4..2]被解释为(a3&b4,a2&b3,a1&b2)，如(a,b,c)&(d,e,f)被解释为(a&d,b&e,c&f)。

③ 如果一个操作数为单独节点或常量 VCC,GND，另一个操作数为节点组，那么它们的逻辑运算可认为是单独节点或常量对节点组中的每个成员分别进行逻辑运算。a&b[3..1]为(a&b3,a&b2,a&b1)。

④ 如果两个操作数都是数值，那么短的数值将带符号扩展至另一个操作数的长度，然后按位运算。如 3#8，先将数值变为 B"0011"#B"1000"，其结果为 B"1011"。

⑤ 如果一个操作数是数值而另一个是节点组，那么这个数值将被裁至或带符号扩展至节

点组的长度，然后按位运算。

表 9.2.3 逻辑运算符

运算符	例 子	说 明
! NOT	!a NOT a	“非”
& AND	Bread & butter bread AND butter	“与”
!& NAND	a[3..1] !& b[5..3] a[3..1] NAND b[5..3]	“与非”
# OR	trick # treat trick OR treat	“或”
!# NOR	c[8..5] !# d[7..4] c[8..5] NOR d[7..4]	“或非”
$ XOR	foo $ bar foo XOR bar	“异或”
!$ XNOR	x2 !$ x4 x2 XNOR x4	“异或非”(同门)

6. 逻辑表达式

逻辑表达式是由操作数以及它们之间的逻辑运算符、算术运算符、比较符和括号组成的。逻辑表达式常用于逻辑等式、Case 和 IF 语句中。逻辑表达式有如下形式：

① 一个操作数，如 a,b[3..1],6,VCC ；

② 一个内部逻辑函数引用，如 out[15..0]=16dmux(q[3..0])；

③ 在逻辑表达式前面加一个前缀运算符(! 或－)；

④ 逻辑表达式括在括号中，如(!foo & bar)；

⑤ 两个表达式之间夹一个二元运算符。

逻辑表达式运算结果与它的操作数的宽度是相同的。

在逻辑表达式中运算符的优先级按由高到低的顺序为：-(一元)、!（非)、算术运算符、比较符、逻辑运算符。相同优先级将按从左至右运算，圆括号()可改变运算顺序。

9.2.2 AHDL 文件的基本结构

用 AHDL 语言编写的设计文件(源文件. tdf)是一个 ASCII 文本文件。文件结构如图 9.2.1 所示。AHDL 文件由许多语句和三个段组成。一个 TDF 文件必须有一个子设计段和一个逻辑段。可以有选择地包括一个变量段、选择语句、标题语句和默认语句，以及一个或多个包含语句、常量语句、定义语句和函数原型语句。子设计段、变量段和逻辑段形成了 TDF 文件的行为描述，是设计文件的主要内容。

在一个设计层次里的文件可以是 TDF 文件、VHDL 文件、GDF 文件、WDF 文件、ADF 文件、SMF 文件、EDIF 文件、ORCAD 图形文件(. sch)、或 XILINX 网表格式文件。通过每个逻

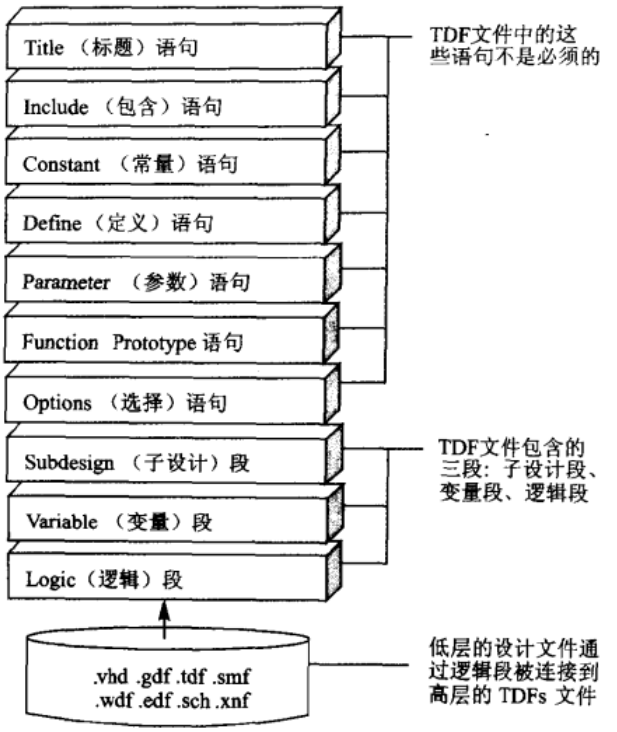


图 9.2.1　AHDL 的文件结构

辑函数的输入和输出端口，把它们同更高层次的设计文件相连接。

下面分别介绍子设计段、逻辑段和变量段以及利用 AHDL 模板建立 TDF 文件。

1. 子设计段

子设计段用于说明逻辑设计的输入、输出和双向端口。描述该设计文件输入/输出端口的类型。

端口的类型通常为以下几种：INPUT（输入）、OUTPUT（输出）、BIDIR（双向）、MACHINE INPUT(状态机输入)、MACHINE OUTPUT(状态机输出)。状态机输入/输出端口仅用于文件之间输入和输出，不能用在顶层文件中，因为顶层文件的端口将对应器件芯片的引脚。

子设计段的格式为

```
SUBDESIGN　子设计名
(
  输入端口：端口类型；
        …
  输出端口：端口类型；
        …
)
```

子设计段是由关键字 SUBDESIGN 开头，后跟子设计名(与 TDF 文件名相同)，然后用圆括号括引输入/输出端口的类型说明。端口的类型说明由冒号(:)把端口(多个端口用逗号分开)和端口类型分开，并用一个分号结束。下面给出一个子设计段的例子。

```
SUBDESIGN  top
(
    foo,bar,clkl,clk2      ;INPUT=VCC;     -- 输入端口默认值为 VCC
    ao,al,a2,a3,a4         ;OUTPUT;        -- 输出端口
    b[7..0]                ;BIDIR ;        -- 双向端口
)
```

2. 逻辑段

逻辑段是用来描述逻辑电路的功能，编写 TDF 文件的逻辑操作，是一个 TDF 文件的主体。逻辑功能的描述用语句来表示，常用的语句有

- 逻辑等式；
- 逻辑控制等式；
- Case(情况)语句；
- 函数内部直接引用语句；
- If 语句；
- Defaults(默认)语句；
- 真值表语句；
- For 语句。

逻辑段由关键字 BEGIN 开头，END 结尾，紧跟一个分号(;)。中间由以上这些描述逻辑功能的语言组成。

逻辑段的格式如下：

```
BEGIN
    语句 ;
    ...
END ;
```

例如：

```
SUBDESIGN boole1
(
    a0, a1, b          ;INPUT ;
    out1, out2         ;OUTPUT;
)
BEGIN
    out1=a1 & !a0;
    out2=out1 # b;
END ;
```

AHDL 是一种并行语言，在一个 TDF 文件的逻辑段中定义的所有行为都是在同一时间内并行运算、操作，而不是顺序进行的。对于同一个节点和变量多次赋值的语句之间形成逻辑连接，如果节点和变量是高电平有效(默认值为 GND)，那么这些语句之间是“或”关系；如果节点和变量是低电平有效(默认值为 VCC)，那么语句之间是“与”关系。

3. 变量段

变量段(Variable Section)用于说明和产生用在逻辑段的任何变量(符号名)，AHDL 的变量类似于在高级编程语言中的变量，用来定义内部的逻辑(表示内部节点的连接关系)。

变量段的格式为

```
VARIABLE
    变量名:变量类型;
```

…

变量段由关键字 VARIABLE 开始，定义的各变量名之间用逗号分隔，变量名与变量类型之间以冒号分开，最后用分号结束。这样完成了定义内部逻辑变量的一个语句。变量类型可以是节点说明、实例说明、寄存器说明或状态机说明。例如：

```
SUBDESIGN boole2
(
   a0, a1, b      :INPUT;
   out            :OUTPUT;
)
VARIABLE
   a_equals_2     :NODE;
BEGIN
   a_equals_2=a1 &!a0;
   out=a_equals_2 # b;
END;
```

(1) 节点说明

AHDL 支持两种类型的节点：**NODE**(节点)和 **TRI_STATE_NODE**(三态节点)。它们是一个全能的变量类型，用来存储在子设计段中没有被说明过的信号。因此这个变量可以被用在一个等式的左边或右边。

节点和三态节点类似于子设计段的输入、输出和双向端口类型，也代表传输信号的一条信号线。

(2) 实例说明

实例(Instance)是对系统已定义的元件模块、函数模块和用户自建的功能模块的统称。它可以被一个 TDF 文件的逻辑段多次引用。在引用之前，可通过变量段中的实例说明，对要引用的元件模块、函数和功能模块指定模块名(变量名)。在逻辑段中通过模块名的引用，实现该模块的功能。

在逻辑段中，一个已被说明的实例(已定义了变量名)需要与其他逻辑信号相连，可按下列格式来表示一个实例端口：

(模块)变量·端口名

例如，希望把一个叫 compare 的函数调入当前的 TDF 文件中，就应该在变量段中作如下的实例说明：

```
VARIABL
   b:compare ;
```

变量名 b 是函数 compare 的一个实例名，它如果有以下端口：

```
a[3..0],c[4..0]   :INPUT;
out1 ,out2        :OUTPUT;
```

那么在逻辑段中，实例名 b 的端口可以表示为：b. a[]，b. c[]，b. out1，b. out2。这些端口可以和节点一样在任何行为语句中使用。

(3) 寄存器说明

寄存器说明包括 D，T，JK，SR 触发器(即 DFF，DFFE，TFF，TFFE，JKFF，JKFFE，SRFF

和 SRFFE)和锁存器(LATCH)的说明。这些寄存器已被系统预先定义,可直接引用,其说明格式与实例说明完全相同。例如:

```
SUBDESIGN bur_reg
(
    clk, load, d[7..0]   ;INPUT;
    q[7..0]              ;OUTPUT;
)
VARIABLE
    ff[7..0]             ;DFFE          ;% ff[7..0]定义 8 个 D 触发器名 %
BEGIN
    ff[].clk=clk;
    ff[].ena=load;
    ff[].d=d[];
    q[]=ff[].q;
END;
```

在这个例子中实例为 D 触发器(DFFE),用组 ff[7..0]来表示 8 个 DFFE 的变量名(寄存器说明),其触发器端口为:ff[].clk,ff[].ena, ff[].d,ff[].q。

在 MAX+PLUS II 系统中已预先定义了触发器的端口名,在 TDF 文件中不需要说明,可直接引用端口名。通常使用的触发器的端口名列于表 9.2.4 中。

表 9.2.4 触发器的端口

端口名	说 明
.q	一个触发器或锁存器的输出端
.d	一个 D 触发器或锁存器的数据输入端
.t	T 触发器的数据输入端
.j .k	JK 触发器的数据输入端
.s .r	SR 触发器的设置输入端 SR 触发器的清除输入端
.clk	触发器的时钟输入端
.ena	触发器、锁存器、状态机的使能输入端
.prn	触发器的低电平有效异步置 1 输入端
.clrn	触发器的低电平有效异步置 0 输入端

(4) 状态机说明

要创建一个状态机,必须在变量段内说明状态机的名称、状态以及状态位。下面给出一个状态机说明的例子:

```
VARIABLE
    ss:MACHINE
      OF BITS(q1,q2,q3)      % 可选 %
```

```
        WITH STATES (
          S1=B"000",
          S2=B"001",
          S3=B"001" );
```

状态机的名称为 ss，由关键字 MACHINE 定义。状态位 q1，q2 和 q3 由关键字 OF BITS 和圆括号来指定(可选)，状态位是状态机寄存器的输出端。状态机的状态 s1，s2 和 s3 由关键字 WITH STATES 和圆括号来说明，每个状态给状态位 q1，q2 和 q3 赋予一个状态值。下面为状态机的一个设计文件。

```
SUBDESIGN recover
(
    clk :INPUT;
    go:INPUT;
    ok:OUTPUT;
)
VARIABLE
    sequence :MACHINE
        OF BITS (q[2..0])
        WITH STATES (
            idle,
            one,
            two,
            three,
            four,
            illegal1,
            illegal2,
            illegal3);
BEGIN
    sequence.clk=clk;
  CASE sequence IS
    WHEN idle=>
        IF go THEN    sequence=one;
        END IF;
    WHEN one      =>sequence=two;
    WHEN two      =>sequence=three;
    WHEN three    =>sequence=four;
    WHEN OTHERS=>sequence=idle;
  END CASE;
  ok=(sequence ==four);
END;
```

在该例中共有 8 个状态，其中，有 5 个有效状态(idle，one，two，three，four)，3 个无效状态(illegal1，illegal2，illegal3)。本例中没有具体指定每个状态对应的编码值(由系统自动指定)。为了保证该设计文件能自启动(使无效状态能进入有效状态)，本例采用了 Case 语句中的

WHEN OTHERS 语句，把无效状态强制转换为有效状态 idle。

4. AHDL 模板

为了设计者方便快捷地进行设计输入，MAX＋PLUS II 系统提供了 AHDL 模板和 AHDL 例子。AHDL 模板包括各种语句和设计结构。在文本编辑器中，使用 AHDL 模板（选菜单项 Template\AHDL Template...），可以把 AHDL 模板加入到设计输入的 TDF 文件中。一旦插入一个模板（关键字是大写字母，每个变量名以两个下划线开头），设计者必须用自己的逻辑设计取代模块中所有变量或表达式，这样可加速 TDF 文件的设计输入。

MAX+PLUS II 系统提供的 AHDL 例子都存在\max2work\ahdl 目录内，设计者可使用这些例子，并且可根据自己的设计要求编辑这些例子文件。

9.2.3 函数模块及其引用

在设计 AHDL 文件时可利用系统已定义的基本元件、函数和已设计的模块建立源文件。在 TDF 文件的逻辑段中进行逻辑设计调用前，还需对这些基本元件和函数模块进行说明，即函数原型说明和变量名定义（变量段）。

1. 函数原型语句

函数原型语句（Function Prototype Statement）与原理图设计文件中的符号具有相同的功能，二者都作为功能模块使用，为一个逻辑函数关系提供简略的描述，并且包括它的各称、输入、输出端口和双向端口等等。

为了在逻辑段中能调用函数模块，首先必须保证该函数已经在它自己的文件中定义了相应的逻辑功能，然后使用函数原型语句来说明该函数的输入/输出端口，并且可以采用内部直接引用或者实例说明方式来调用该函数模块。

函数原型说明语句必须被放在子设计段的外面。以关键字 FUNCTION 开始，后跟函数名和一组输入信号的端口名（用逗号分开），然后在关键字 RETURNS 后列出的是该函数的一组输出信号端口或双向端口，最后以分号结尾。注意这些输入/输出端口名一定要与已定义的函数模块中所用的端口名相同。如果函数提供的输入/输出端口为状态机端口，那么一定要用关键字 MACHINE（状态机）进行说明。

函数原型语句分为带参数和不带参数的两种格式。

- 带参数格式：

```
FUNCTION 函数名(一组输入端口)
WITH(参数表)
RETURNS(输出端口或双向端口);
```

- 不带参数格式：

```
FUNCTION 函数名(一组输入端口)
RETURNS(输出端口或双向端口);
```

例如：

```
FUNCTION bus_reg2 (clk, oe) RETURNS (io);
SUBDESIGN bidir1
(
   clk, oe    :INPUT;
```

```
    io[3..0] ;BIDIR;
  )
BEGIN
    io0=bus_reg2(clk, oe);
    io1=bus_reg2(clk, oe);
    io2=bus_reg2(clk, oe);
    io3=bus_reg2(clk, oe);
END;
```

在这个例子中，bidir1 设计文件实现了直接调用一个函数名为 bus_reg2 的模块。bus_reg2 函数模块的电路图如图 9.2.2 所示，而对应的 AHDL 文本文件如下：

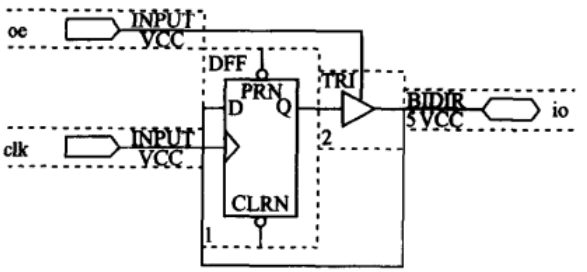


图 9.2.2　bus_reg2 函数模块电路

```
SUBDESIGN bus_reg2
(
    clk     ;INPUT;
    oe      ;INPUT;
    io      ;BIDIR;
  )
BEGIN
     io=TRI(DFF(io, clk, , ), oe);
END;
```

由于触发器(DFF)和三态缓冲器(TRI)是被系统预先定义好的函数模块(默认函数)，故设计者可以直接引用，而不必在设计文件中进行函数原型语句说明，但使用的端口与函数模块端口名的位置应该一一对应。

在一个 TDF 文件中使用函数原型说明语句的另一个方法是：用一个 Include(包含)语句代替。

2. 包含语句

包含语句(Include Statement)允许设计者从一个包含文件(.inc)向当前文件引入文本。编译器在对设计文件进行处理时，将包含文件(.inc)中的文本内容替代调用这个文件的包含语句。包含文件内容可以为函数原型语句、定义语句、参数和常量语句，不能包含子设计段。

包含语句的格式如下：

INCLUDE"文件名";

例如：

```
INCLUDE"const.inc";
```

在编译时，包含文件 const. inc 将替代该包含语句(INCLUDE"const. inc";)。

系统编译器将按照下列顺序查找包含文件所在的目录：

① 设计文件的当前目录；

② 使用 User Libraries(Option menu)指定用户库；

③ 在系统安装时建立的\maxplus2\max2lib\mega - lpm(强函数的包含文件)和\maxplus2\max2inc(宏函数的包含文件)目录。

MAX+PLUS II 提供了 Greater Default Include File 命令(File menu)，用该命令可以为任何设计文件(图形文件、波形文件、文本文件)自动建立一个包含文件(*.inc)。

3. 基本元件

MAX+PLUS II 系统为原理图设计提供了各种基本元件(Primitive 原语)(在\maxplus2\max2lib\prim 目录中)，AHDL 语言所使用的基本元件(原语)只是电路设计中的一部分。原理图设计文件中的某些基本元件在 AHDL 文件中被一些运算符、关键字和语句所代替。

AHDL 文件中的基本元件主要是缓冲器、触发器和锁存器。TDF 文件在调用基本元件模块时，无需进行函数原型语句说明或用 Include 语句说明要调用的这些模块，因为这些基本元件已被系统预先定义和默认(用保留标识符表示)。

(1) 触发器和锁存器

表 9.2.5 列出了 MAX+PLUS II 触发器和锁存器的函数原型。所有触发器都是上升沿触发，而锁存器是电平触发。当锁存器使能或触发器时钟使能(ena)输入信号为高电平时，触发器或锁存器会把数据输入端的信号传到输出端(q)。当使能端为低电平时，输出端状态将保持，数据输入端无法输入信号。

表 9.2.5 MAX+PLUS II 触发器和锁存器的函数原型

触发器/锁存器	AHDL 函数原型语句	
LATCH	FUNCTION latch (d,ena)	RETURNS(q);
DFF	FUNCTION dff(d,clk,clrn,prn)	RETURNS(q);
DFFE	FUNCTION dffe(d,clk,clrn,prn,ena)	RETURNS(q);
JKFF	FUNCTION jkff(j,k,clk,clrn,prn)	RETURNS(q);
JKFFE	FUNCTION jkffe(j,k,clk,clrn,prn,ena)	RETURNS(q);
TFF	FUNCTION tff(t,clk,clrn,prn)	RETURNS(q);
TFFE	FUNCTION tffe(t,clk,clrn,prn,ena)	RETURNS(q);
SRFF	FUNCTION srff(s,r,clk,clrn,prn)	RETURNS(q);
SRFFE	FUNCTION srffe(s,r,clk,clrn,prn,ena)	RETURNS(q);

注：clk=时钟输入端；clrn=清零输入端；d,j,k,t,s,r=数据输入端；ena=锁存器使能或触发器时钟使能输入端；prn=预置输入端；q=输出端。

由于所有触发器和锁存器都只有一个输出端口，因此想使用其输出端口，可以直接在等式右边使用该模块的变量而不必带上端口名。同样对于只有一个输入端的函数模块(如 DFF，DFFE，TFF，TFFE)，可以在等式左边直接使用该函数模块名。例如下面文件中有：

```
VARIABLE
    a ,b :DFF;
BEGIN
    a=b;
END;
```

在这里的逻辑段中 a=b(不带端口名)与 a.d=b.q 的含义是相同的。

(2) 缓冲器

AHDL 提供的缓冲器有:CARRY(进位缓冲器)、CASCADE(级联缓冲器)、EXP(扩展缓冲器)、GLOBAL(全局缓冲器)、LCELL(逻辑单元缓冲器)、MCELL(宏单元缓冲器)、OPENDRN(漏极开路缓冲器)、SCLK(同步时钟缓冲器)、SOFT(放缓冲器)、TRI(三态缓冲器)。

由于缓冲器元件仅用于对逻辑综合过程进行控制(有效地利用硬件资源),而不是用于逻辑设计,因此在多数情况下,不需要使用这些缓冲器元件(TRI 除外),尤其对初学者,不必花费时间去理解和使用这些元件。但是如果编译器提示所作的设计太复杂而无法处理时,那么设计者可以试着在设计中插入上述某些缓冲器,以引导逻辑综合器产生所期望的结果。

重点介绍三态缓冲器 TRI,逻辑符号如图 9.2.3 所示,其函数原型如下:

```
FUNCTION tri (in, oe)
    RETURNS (out);
```

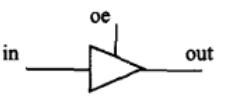


图 9.2.3　TRI 三态缓冲器

当 oe=1 时,TRI 允许输入信号直接从输出端输出,即 out=in;

当 oe=0 时,TRI 的输入信号无论为何值,输出端均呈现高阻态。

在使用 TRI 三态缓冲器时,应遵守以下规则。

● 一个 TRI 缓冲器只能驱动一个双向引脚(BIDIR 或 BIDIRC),但可以驱动多个输出引脚(OUTPUT 或 OUTPUTC)。

● 如果在 TRI 输出端有自反馈,那么 TRI 输出端必须连接一个双向引脚(BIDIR 或 BIDIRC)。

● 在输出使能信号端(oe)不恒为高电平(VCC)时,TRI 输出一定要连接到输出引脚端或双向引脚端上,因为内部信号不可以为高阻状态。

4. 强函数和宏函数模块

除了基本元件模块外,MAX+PLUS II 还提供了一系列结构更复杂,功能更强大的函数模块:强函数和宏函数。这些函数模块是已设计好的实例,在调用这些函数模块时,首先应该进行函数原型语句说明或 Include 语句说明。

(1) 强函数

强函数(Megafunction)是一种复杂的逻辑函数的集合。它包括参数设置模式的库函数(LPM),可在逻辑设计中引用。强函数被系统自动地安装在\maxplus2\max2lib\mega - lpm 目录中,这些目录也包括了具有每种强函数的函数原型语句(带参数)的包含文件(.inc)。

在所有 MAX+PLUS II 逻辑设计中用户可以灵活地应用这些强函数。当编译器分析一个逻辑电路时,自动地移去所有不用的门和触发器,使这种设计效率不被降低。

包含 LPM 函数的强函数名字用"lpm_"起头。可分为:门函数模块、运算部件模块、存储

部件模块和其他功能模块，如表 9.2.6 所列。关于这些强函数的详细信息可利用 MAX+PLUS II 的“帮助”获得。

表 9.2.6 MAX+PLUS 的强函数

类 型	各 称	说 明
门 (Gates)	lpm_and	参数设置的“与”门
	lpm_or	参数设置的“或”门
	lpm_inv	参数设置的反相器
	lpm_xor	参数设置的“异或”门
	lpm_mux	参数设置的多路选择器
	lpm_decode	参数设置的译码器模块
	lpm_bustri	参数设置的三态缓冲器
	lpm_clshift	参数设置的组合移位模块
	lpm_constant	参数设置的恒定振荡器模块
运算部件 (Arithmetic Components)	lpm_abs	参数设置的绝对值
	lpm_add_sub	参数设置的加法/减法模块
	lpm_decode	参数设置的比较器模块
	lpm_counter	参数设置的计数器模块
	lpm_mult	参数设置的相乘器模块
存储部件 (Storage Components)	Lpm_dff	参数设置的 D 触发器和移位寄存器
	Lpm_latch	参数设置的锁存器组件
	lpm_ram_dg	具有独立输入和输出端口的随机存取存储器
	lpm_ ram_io	具有一个单 I/O 端口的随机存取存储器
	lpm_rom	只读存储器
	lpm_tff	参数设置的 T 触发器组件
	csdpram	循环分配双端口随机存取存储器
	csfifo	循环分配先进先出
其他功能	ntsc	NTSC 视频控制信号发生器
	pll	上升沿和下降沿检测器

(2) 宏函数

宏函数(Macrofunction)是具有一定功能的函数模块，对常用的逻辑功能已设计成为一个函数模块。其中包括 74 系列数字集成电路的逻辑功能，为设计数字系统提供方便。

宏函数被系统自动地安装在\maxplus2\max2lib 目录及其子目录中。包含宏函数的函数原型语句(不带参数)的 Include 文件被系统安装在\maxplus2\max2inc 目录中。设计者在使用这些宏函数时，需要用函数原型语句或包含语句进行说明，然后可采用内部直接引用或者一种实例说明。

表 9.2.7 列出了 MAX+PLUS II 的主要宏函数。有关宏函数的详细信息请参阅 MAX+PLUS II 的帮助信息。

表 9.2.7　MAX+PLUS II 的宏函数

宏函数类型	宏函数名称	说　明
Adder 加法器	8fadd 7482 7483 74183 74283 74385	8 位全加器 2 位二进制数全加器 快速进位 4 位全加器 双进位保留全加器 4 位二进制超前进位全加器 带清零端 4bit 加/减法器
Arithmetic Logic Units 算术逻辑单元	74181 74182 74381 74382	4 位算术逻辑单元/函数(功能)发生器 超前进位产生器 4 位算术逻辑单元 超前进位产生器
Application Specific 专用	ntsc pll	NTSC 视频控制信号发生器 上升沿或下降沿检测器
Buffer 缓冲器(三态)	74240 74244 74365 74366 74367 74368 74465 74466 74468 74541	八反相缓冲器/线驱动器 八同相缓冲器/总线驱动器 六同相缓冲器/线驱动器 六反相缓冲器/线驱动器 六同相缓冲器/线驱动器 六反相缓冲器/线驱动器 八总线缓冲器 八反相缓冲器 八反相缓冲器 八总线缓冲器
Comparator 比较器	8mcomp 7485 74518 74684 74686 74688	8 位数值比较器 4 位数值比较器 8 位等值检测器 8 位数值/等值比较器 8 位数值/等值比较器 8 位等值检测器
Converters 转换器	74184 74185	BCD-二进制码转换器 二进制码-BCD 转换器

续表 9.2.7

宏函数类型	宏函数名称	说 明
Counter 计数器	gray4	格雷码计数器
	unicnt	全能 4 位左/右移、加/减计数器(异步清除、并入)
	16cudslr	16 位左/右移、加/减计数器(异步预置、清除)
	4count	4 位加/减计数器(并行输入、异步清除)
	8count	8 位加/减计数器(同步并行输入)
	7468	双十进制计数器(BCD 码输出)
	7469	双 4 位二进制计数器
	7490	二-五-十进制计数器
	7492	十二进制计数器
	7493	4 位二进制计数器
	74160	十进制计数器(同步并入、异步清除)
	74161	4 位二进制计数器(同步并入、异步清除)
	74162	十进制计数器(同步并入、清除)
	74163	4 位二进制计数器(同步并入、清除)
	74168	同步十进制加/减计数器
	74169	同步 4 位二进制加/减计数器
	74176	可预置十进制计数器
	74177	可预置 4 位二进制计数器
	74190	同步十进制加/减计数器(异步并入)
	74191	同步 4 位二进制加/减计数器(异步并入)
	74192	同步十进制加/减计数器(异步清除)
	74193	同步 4 位二进制加/减计数器(异步清除)
	74196	二-五-十进制计数器(可预置)
	74197	二-八-十六进制计数器(可预置)
	74290	二-五-十进制计数器
	74292	可编程分频器/数字定时器
	74293	二-八-十六进制计数器
	74294	可编程分频器/数字定时器
	74390	双十进制计数器
	74393	双 4 位二进制计数器(异步清除)
	74490	双十进制计数器
	74568	十进制加/减计数器(异步并入)
	74569	4 位二进制加/减计数器(异步并入)
	74590	8 位二进制计数器(带三态输出)
	74592	8 位二进制计数器(可预置)
	74668	同步十进制加/减计数器
	74669	同步 4 位二进制加/减计数器
	74690	同步十进制计数器(3s、异步清除)
	74691	同步 4 位二进制计数器(3s、异步清除)
	74693	同步 4 位二进制计数器(3s、同步清除)
	74696	同步十进制加/减计数器(3s、异步清除)
	74697	同步 4 位二进制加/减计数器(3s、异步清除)
	74698	同步十进制加/减计数器(3s、同步清除)
	74699	同步 4 位二进制加/减计数器(3s、同步清除)

续表 9.2.7

宏函数类型	宏函数名称	说　明
Decoder 译码器	16dmux	4 线-16 线译码器
	7442	4 线-10 线译码器(BCD 码输入)
	7445	BCD-十进制译码器
	7447	4 线-七段译码器/驱动器
	7448	4 线-七段译码器/驱动器
	7449	4 线-七段译码器/驱动器
	74137	3 线-8 线译码器(带地址锁存)
	74138	3 线-8 线译码器
	74139	双 2 线-4 线译码器
	74154	4 线-16 线译码器
	74247	BCD-七段译码器
	74248	BCD-七段译码器
	74445	BCD-十进制译码器
Digital Filter 数字滤波器	74297	数字锁相环滤波器
Encoder 编码器	74147	10 线-4 线编码器(BCD 码输出)
	74148	8 线-3 线编码器
	74348	8 线-3 线优先编码器(3s)
Latch 锁存器	explatch	D 锁存器
	norltch	SR 锁存器
	7475	4 位双稳态锁存器
	74116	双 4 位锁存器(清除)
	74259	8 位可寻址锁存器/3 线-8 线译码器
	74373	8 位 D 锁存器(3S)
	74604	8 位 2 选 1 选择器/锁存器(3S)
	74841	10 位 D 锁存器(3S)
	74843	9 位总线 D 锁存器(3S)
	74846	8 位总线 D 锁存器(3S)
	74990	8 位读取锁存器
Multiplier 乘法器	mult2	2 位乘法器
	mult24 mult4	4×2 位二进制乘法器
	74261	4 位二进制乘法器
	74284	2 位二进制乘法器
	74285	4 位二进制乘法器(结果取高四位)
		4 位二进制乘法器(结果取低四位)

续表 9.2.7

宏函数类型	宏函数名称	说　明
Multiplexer 多路选择器	21mux	2 选 1 多路选择器
	81mux	8 选 1 多路选择器
	161mux	16 选 1 多路选择器
	74151	8 选 1 多路选择器
	74153	双 4 选 1 多路选择器
	74251	8 选 1 数据选择器(3S)
	74253	双 4 选 1 数据选择器(3S)
	74352	双 4 选 1 数据选择器(输出取反)
	74354	8 选 1 数据选择器(3S)
	74398	2 选 1 - 4 输入转换器(存储)
Register 寄存器	7470	与门输入 JK 触发器(预置、清零)
	7471	与或门输入 JK 触发器
	7474	双 D 触发器(异步预置、清零)
	7476	双 JK 触发器(异步预置、清零)
	74107	双 JK 触发器
	74112	双 JK 触发器(下降沿触发)
	74171	4 位 D 触发器(清零)
	74174	6 位 D 触发器(清零)
	74175	4 位 D 触发器(清零)
	74273	4 位 JK 触发器(预置、清零)
	74374	8 位 D 触发器(3S)
	74377	8 位 D 触发器(带使能端)
	74378	6 位 D 触发器(带使能端)
	74396	8 位存储器/寄存器
	74548	8 位×2 寄存器(3S)
	74821	10 位总线触发器(3S)
	74823	9 位总线触发器(3S)
	74826	9 位总线触发器(3S、反相输出)
Shift Register 移位寄存器	barrelst	8 位双向移位寄存器(并入)
	74164	8 位移位寄存器(串行输入、并行输出)
	74165	8 位移位寄存器(并行输入、串行输出)
	74179	4 位并行存取移位寄存器(串行输入)
	74194	4 位双向移位寄存器(并入)
	74195	4 位移位寄存器(并行存取、串行输入)
	74198	8 位双向移位寄存器(并行存取)
	74199	8 位移位寄存器(并行存取、串行输入)
	74299	8 位双向通用移位寄存器
	74589	8 位输入锁存、串行输出移位寄存器(3S)
	74595	8 位移位寄存器(3S)
	74671	4 位通用移位寄存器/锁存器(3S)
	74674	16 位移位寄存器(3S)

续表 9.2.7

宏函数类型	宏函数名称	说　明
Storage Register 存储寄存器	7498 74278	4 位×2 数据选择器/存储器 4 位可级联优先寄存器
SSI Functions 小规模功能电路	7400	NAND2 Gate
	7402	NOR2 Gate
	7404	NOT Gate
	7408	AND2 Gate
	7410	NAND3 Gate
	7411	AND3 Gate
	7420	NAND4 Gate
	7421	AND4 Gate
	7423	Dual 4 - Input NOR Gate with Strobe
	7425	Dual 4 - Input NOR Gate With Strobe
	7427	NOR3 Gate
	7428	Quad 2 - Input Positive NOR Buffer
	7430	NAND8 Gate
	7432	OR2 Gate
	7437	Quad 2 - Input Positive NAND Buffer
	7440	Dual 4 - Input Positive NAND Buffer
	7450	Dual 2 - Wide 2 - Input AND - OR - INVERT Gate
	7451	Dual AND - OR - INVERT Gate
	7452	AND - OR Gate
	7453	Expandable 4 - Wide AND - OR - INVERT Gate
	7454	4 - Wide AND - OR - INVERT Gate
	7455	2 - Wide, 4 - Input AND - OR - INVERT Gate
	7464	4 - 2 - 3 - 2 - Input AND - OR - INVERT Gate
	7486	XOR Gate
	74133	13 - Input NAND Gate
	74134	2 - Input NAND Gate with Tri - State Output
	74135	Quad XOR/XNOR Gates
	74260	Dual 5 - Input Positive NOR Gates
	74386	Quad XOR Gate

9.2.4　AHDL 的描述语句

1. 逻辑等式(赋值语句)

在 TDF 文件逻辑段中的逻辑等式是用来表示节点之间的连接以及输入/输出引脚、函数模块和状态机的输入信号流和输出信号流。

在逻辑等式中，用一个等号符号“=”来表示等式右边逻辑表达式的结果将赋给左边的符号(变量)节点或组。等式左边可以是一个符号(变量)、端口或组名，还可以把 NOT(即!)运算符对左边任何项求反。逻辑等式在使用时应符合以下规则：

① 同一个变量的多次赋值之间在逻辑上是“或”的关系，除非该变量的默认值为 VCC。

② 如果等式两边组的长度相同，那么两边组的节点一一对应赋值。如果两边组的长度不相同，那么左边组的位数一定要能被右边组的位数整除。

③ 如果一个单独的节点、VCC 或 GND 被赋予一个组，那么把这个节点或常量赋值给这个组中的每个成员。

④ 在逻辑等式中逗号(,)可以用来保留没有被赋值的成员位置。如：(a , ,c ,) = B “1011”；表示 a 和 b 被赋值为 1。

⑤ 每个等式都以分号(;)结束。

2. 真值表

真值表格式如下：

```
TABLE
    节点名,节点名=>节点名,节点名;
    输入值,输入值=>输出值,输出值;
                ...
    输入值,输入值=>输出值,输出值;
END TABLE;
```

真值表表头由关键字 TABLE、一组由逗号和箭头符号(=>)分开的真值表输入和输出的节点名组成，并由一个分号(;)结束。表中的每项包含输入值的一种组合形式以及所产生的逻辑输出值。输入和输出值对应表头的输入、输出节点。输入和输出值可以是数值、常量 VCC 或 GND、符号常量，输入值还可以是 X(无关项)。例如：

```
TABLE
  a0,     f[4..1].q    =>    f[4..1].d,     control;
  0,       B"0000"     =>     B"0001",      1;
  0,       B"0100"     =>     B"0010",      0;
  1,       B"0XXX"     =>     B"0100",      0;
  X,       B"1111"     =>     B"0101",      1;
END TABLE;
```

3. Case 语句

Case 语句(情况语句)列出了几种可能执行的操作，实际执行什么操作，取决于关键字 CASE 后面的变量、组或表达式的值。Case 语句的格式如下：

```
CASE   变量、组或表达式 IS
    WHEN 常数值  =>
        语句;
    WHEN 常数值  =>
        语句;
    WHEN OTHERS=>
        语句;
```

```
END CASE;
```

Case 语句是逻辑段中一个经常使用的语句。它以关键字 CASE 开始，以 END CASE 结束。关键字 WHEN 后跟的常数值为变量、组或表达式的可能取值，当变量、组或表达式满足这个常数值时，就执行后面的语句。如果变量、组或表达式都不满足所有 WHEN 后跟的常数值，那么就执行关键字 WHEN OTHERS 后跟的语句或者跳出 Case 语句。例如：

```
SUBDESIGN 4p
(
        a[3..0]:INPUT ;
        out     :OUTPUT;
)
BEGIN
  CASE a[] IS
        WHEN B"x111" =>     out=VCC;
        WHEN B"1x11" =>     out=VCC;
        WHEN B"11x1" =>     out=VCC;
        WHEN B"111x" =>     out=VCC;
        WHEN OTHERS=>       out=GND;
  END CASE;
END;
```

4. IF Then 语句

IF Then 语句工作过程流程图如图 9.2.4 所示。

IF Then 语句的格式如下：

```
IF 表达式 1 THEN
  语句 1;
ELSIF 表达式 2 THEN
  语句 2;
ELSE
  语句 3;
END IF;
```

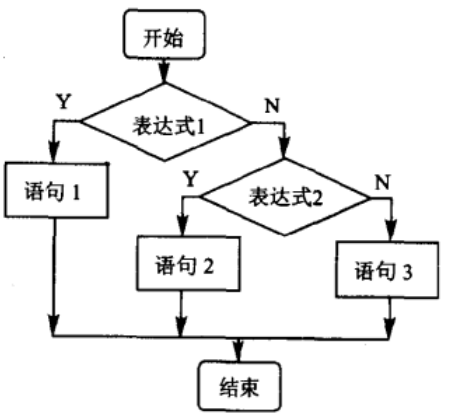


图 9.2.4　IF Then 语句流程图

在 IF Then 语句中可以有一个或多个表达式。如果其中某表达式结果为真，那么该表达式后面的行为语言将被执行。如果所有表达式结果都不为真，那么就执行 ELSE 后面的语句（可选）。

例如下面是用 AHDL 语言构成的 16 位二进制计数器（清零、并入）源文件：

```
SUBDESIGN ahdlcnt
(
   clk, load, ena, clr, d[15..0] :INPUT;
   q[15..0]                      :OUTPUT;
)
VARIABLE
```

```
    count[15..0]        :DFF;

BEGIN
    count[].clk=clk;
    count[].clrn=!clr;
  IF load THEN                          --load 为高电平时,数据同步并入
      count[].d=d[];
   ELSIF ena THEN                       --ena 使能端
      count[].d=count[].q+1;
   ELSE
      count[].d=count[].q;              --ena 为低电平时,保持
   END IF;
    q[]=count[];
END;
```

该计数器输入端口有:时钟 clk、并入控制 load、使能端 ena、异步清零 clr、16 位数据输入 d[15..0];输出端口为 q[15..0]。其实现的功能见表 9.2.8(功能表)。

表 9.2.8 16 位二进制计数器功能表

输 入				输 出
clk	clr	load	ena	q[15..0]
×	1	×	×	0...0
↑	0	1	×	d15...d0
↑	0	0	1	加 1 计数
↑	0	0	0	q15...q0

5. If Generate 语句

If Generate 语句与 If Then 语句相似,列出一系列行为语句是在确认运算表达式的运算之后进行执行。其格式如下:

```
IF 表达式 Generate
  语句 1;
  语句 2;
ELSE GENERATE
  语句 3;
  语句 4;
END  GENERATE;
```

当运算表达式为"真"时,则执行语句 1 和语句 2;当运算表达式为"假"时,则执行语句 3 和语句 4。If Generate 语句与 If Then 语句的不同之处有以下几点:

① If Then 语句只能计算逻辑表达式,而 If Generate 语句可以计算算术表达式;

② If Generate 语句可以用在逻辑段或变量段中;

③ If Generate 语句同 For Generate 语句特别有助于以不同方式处理特殊情况。

6. For Generate 语句

For Generate 语句可重复运行其后的执行语句。其格式如下：

```
FOR 变量名  IN 表达式 1  TO 表达式 2  GENERATE
    语句;
END GENERATE;
```

在 For 和 Generate 之间的表达式表示变量名在重复执行语句时的取值范围(表达式 1 到表达式 2)，该变量名是临时的，只用于 For Generate 语句的范围内，在编译器处理这种语句之后该变量名中止存在，所以该变量名可以不先说明。例如：

```
    CONSTANT NUM_OF_ADDERS=8;
    SUBDESIGN 4gentst
    (
    a[NUM_OF_ADDERS..1], b[NUM_OF_ADDERS..1], cin :INPUT;
    c[NUM_OF_ADDERS..1], cout                     :OUTPUT;
      )
    VARIABLE
    carry_out[(NUM_OF_ADDERS+1)..1]               :NODE;
BEGIN
    carry_out[1]=cin;
    FOR i IN 1 TO NUM_OF_ADDERS GENERATE
        c[i]=a[i]$b[i]$carry_out[i];        % Full Adder %
        carry_out[i+1]=a[i]&b[i]#carry_out[i]&(a[i]$b[i]);
    END GENERATE;
    cout=carry_out[NUM_OF_ADDERS+1];
END;
```

7. Defaults 语句(默认语句)

Defaults 语句用来给 IF 语句、Case 语句和真值表语句中的变量设定默认值。对于高电平有效的信号被系统自动设定默认值为 GND(低电平)，所以只有对低电平有效的信号需要用 Defaults 语句设定默认值(高电平 VCC)。

Defaults 语句的格式如下：

```
DEFAULTS
    节点名(变量)=常数值 ;
END DEFAULTS ;
```

Defaults 语句以关键字 DEFAULTS 开始，以 END DEFAULTS 结束，夹在中间的是逻辑等式，用来为节点、组和变量设定默认值数值(高或低电平)。例如：

```
    SUBDESIGN default1
(
        i[3..0]             :INPUT;
        Ascii_code[7..0]    :OUTPUT;
    )
BEGIN
    DEFAULTS
```

```
        Ascii_code[]=B"00111111";              % "?" %
    END DEFAULTS;
    TABLE
        i[3..0]     =>    ascii_code[];
        B"1000"     =>    B"01100001";         % "a" %
        B"0100"     =>    B"01100010";         % "b" %
        B"0010"     =>    B"01100011";         % "c" %
        B"0001"     =>    B"01100100";         % "d" %
    END TABLE;
END;
```

在这个例子中，如果输入端 i[3..0]的信号不是真值表中的四个输入组合(B"1000", B"0100",B"0010",B"0001")，那么输出端 Ascii_code[]的信号就是默认语句中设定的常数值(B"00111111")。

在使用 Defaults 语句时应遵守以下规则：

① 在逻辑段中只能有一个 Defaults 语句，并且它是逻辑段关键字 BEGIN 之后的第一个语句；

② 在默认语句中为一个变量多次赋值，那么只有最后一次赋值有效；

③ 默认语句不能给变量设置一个带 x(无关)位的默认值；

④ 对于多次赋值为低电平有效的变量应该被赋以默认值 VCC。

8. 常量(Constant)语句

常量语句的作用是用一个有意义的符号名来代替一个数值或一个算术表达式(常数)。其格式如下：

CISTANT 符号名=数值或表达式；

常量语句是由关键字 CISTANT 开始，后跟符号名、等号和数值。常量语句必须放在所有 AHDL 段的外边。例如：

```
CONSTANT IO_ADDRESS=H"0370";
  SUBDESIGN decode2
(
        a[15..0]     ;INPUT;
        ce           ;OUTPUT;
    )
BEGIN
        ce=(a[15..0] ==IO_ADDRESS);
END;
```

这个例子中用符号名 IO_ADDRESS 代表十六进制数 H"0370"。用符号名代替数值的方式有利于设计文件的可读性，同时易于修改。

9. 参数(Parameters)语句

参数语句用来说明一个或多个参数，这些参数控制一个参数化的强函数或宏函数的执行。也可为每一个参数指定一个默认值(参数值)。参数语句的格式为

```
PARAMETERS
(
```

```
        参数名=参数值，
        参数名，
        参数名
    );
```

参数语句用关键字 PARAMETERS 开始，随后列出一个或多个参数，用逗号(,)分开，并且这些参数括在圆括号中，句末用一个分号。在使用参数前，必须用参数语句进行说明，参数可被赋予参数值(用等式表示)，参数值可为字符串和数值。例如：

```
PARAMETERS
(
    FILENAME="myfile.mif",
    WIDTH,
    AD_WIDTH=8,
    NUMWORDS=2^AD_WIDTH
);
```

10. 选择语句

选择(Options)语句的作用是用来为整个文件中的组设定位的默认顺序。一般组内的成员是按降序排列，例如 a[4..1]的最左边 a4 为最高有效位(MSB)，a1 为最低有效位(LSB)。为了按升序排列，就必须用 Options 语句对最右位(BIT0)进行指定，如 OPTIONS BIT0=MSB；书写格式可为 a[1..4]，否则编译时会产生警告信息。

Options 语句的格式为

```
OPTIONS  BIT0=有效位;
```

其中有效位可设置为 MSB(最高有效位)、LSB(最低有效位)和 ANY。选择语句位于 TDF 文件的子设计段前，如果该 TDF 文件是顶层文件，那么选择语句将作用于整个文件。

11. 标题语句

标题(Title)语句为编译器产生报告文件(.rpt)提供文档注释。其格式如下：

```
TITLE "字符串";
```

使用标题语句时应遵守以下规则：

① 如在标题内需要用双引号标记，就必须使用两个双引号，如

```
TITLE " ""EPM5130"" Display Controller";
```

② 标题语句在一个 TDF 文件中只能使用一次；

③ 标题语句必须放在所有段之外。

9.3　ABEL－HDL 硬件描述语言

硬件描述语言 ABEL－HDL 支持各种行为的输入方式，其中包括逻辑方程式、真值表和状态图。ABEL－HDL 和 Synario 版本的 ABEL－HDL 编译器及其支持的软件通过仿真对 ABEL－HDL 设计进行功能验证，然后在 CPLD/FPGA 器件上实现其设计功能。ABEL－HDL 的设计也能通过标准格式化转换文件转换成其他设计环境。

9.3.1 ABEL－HDL语言的基本元素

1. 标识符和数值

标识符是一组字母数字串，可表示逻辑信号（逻辑变量），也可用来给器件、引脚、节点、数组、宏等命名。标识符的长度不得超过31个字符，且必须以字母或下划线开头。它与输入的字体有关，大写和小写字母将被视为不同的字。例如以下为合法标识符：

ld　out1　Out1　_X56

在ABEL语言中有四种不同的数制表示数值。如'B1010，'O36，'H2EA9分别表示二进制数1010、八进制数36、十六进制数2EA9；十进制的符号是'D，是系统默认的。

2. 关键字

ABEL中的关键字是系统的保留标识符，每个字各有其特定涵义，不可用来对器件、引脚、节点、数组、宏和信号命名。关键字与大写、小写字母无关。其关键字如表9.3.1所示。

表9.3.1 ABEL－HDL中的关键字

关键字	说　明	关键字	说　明
async_reset	异步复位状态描述	case... endcase	条件选择
declarations	定义段	device	器件定义
end	结束	equations	逻辑方程
functional... block	功能模块定义	fuses	熔丝状态定义
goto	无条件转移	interface	功能模块接口定义
istype	属性定义	trace	跟踪选择
library	库引用	macro	宏定义
module	模块定义	node	节点定义
options	控制选项	pin	引脚定义
property	特征定义	state	状态描述
state_diagram	状态图	state_register	状态寄存器说明
sync_reset	同步复位状态描述	test_vectors	测试向量
title	标题	with... endwith	转移方程语句
truth_table	真值表	when... then... else	条件转移（只在方程中使用）
if... then... else	条件转移（只在状态图中使用）		

3. 逻辑常量

在逻辑方程、真值表和测试向量中逻辑常量有1，0或X（任意态），如涉及到器件输出还可能有高阻态和浮动态。还可以用常量定义语句（用等号表示）使常量用标识符（符号常量）来代表。常量可以是数值，也可以是非数值的特殊常量值（见表9.3.2）。

在实际编写源文件时为书写方便，用常量定义语句将H，L定义为1，0（H，L＝1，0）；将.C.，.X.定义为C和X（C，X＝.C.，.X.）；可用大小写方式输入。

表 9.3.2　ABEL 中的特殊常量值

常量值	说　明	常量值	说　明
H	逻辑高电平	L	逻辑低电平
.C.	时钟输入(电平由低—高—低变化)	.K.	时钟输入(电平由高—低—高变化)
.U.	时钟上升沿(由低到高)	.D.	时钟下降沿(由高到低)
.F.	浮动输入或信号输出	.P.	寄存器预装载
.X.	任意值	.Z.	高阻态

4. 运算符

ABEL 语言中的运算符分为三种:逻辑运算符、算术运算符和比较运算符,其优先级别如表 9.3.3 所列。圆括号的优先级最高。

表 9.3.3　ABEL 语言中运算符

运算符	说　明	优先级	运算符	说　明	优先级
−	二进制补码	1	!	非运算	1
*	乘法	2	&	与运算	2
/	无符号整数除	2	#	或运算	3
%	取模	2	$	异或运算	3
>>	右移	2	! $	同或运算	3
<<	左移	2	==	等于	4
+	加法	3	! =	不等于	4
−	减法	3	<	小于	4
>	大于	4	<=	小于等于	4
>=	大于等于	4			

5. 赋值语句

赋值语句用赋值运算符(等号)来表示,把等式右边的表达式、数值和变量赋予等式左边的变量。对门和寄存器的赋值,其性质是不同的,所以赋值语句分为组合赋值和时序赋值两种。

组合赋值表示等号右边的表达式得出结果后无需延时即赋值给输出信号,时序赋值表示等号右边的表达式得出结果后必须延时到与输出相关的时钟的有效边沿到来后,才能赋值给输出信号。对于组合型赋值用等号(=)表示;对于时序型赋值用符号":="表示。例如:

```
F=! (D1&D2&D3);
Q0:=! Q0&D2;
```

ABEL - HDL 用赋值语句将表达式结果赋予一个信号,一个信号也可以多次赋值,其结果为所有赋值语句的"或",而不是最后一次的赋值。这一点与其他高级语言有所不同。

6. 寄存器的描述

对于组合逻辑电路(门电路)用一个或多个逻辑表达式和赋值语句进行描述。但对于寄存器的描述不这样简单。一个寄存器除了输出方程外,还应有时钟方程、复位方程、预置方程等,必须用一组方程才能完全地描述。通常对一个寄存器又只定义一个标识符,所以这些时钟信

号、复位信号、预置信号需要在该寄存器的标识符后面加入后缀表示。如：Q1.d，Q1.clk，Q1.ar，Q1.ap。

ABEL语言中常用的点后缀如表9.3.4所列。它与关键字一样，大写、小写均可。例如：

```
[Q3..Q0].clk=CLK;
[Q3..Q0].ar=CR;
```

表9.3.4 ABEL语言中常用的点后缀

点后缀	含 义	点后缀	含 义
.ap	异步置1	.pin	引脚反馈
.ar	异步复位	.q	触发器状态输出
.clk	边沿触发器的时钟	.r	RS触发器的R输入
.d	D触发器输入	.s	RS触发器的S输入
.fb	寄存器反馈信号	.sp	同步置1
.j	JK触发器的J输入	.sr	同步复位
.k	JK触发器的K输入	.t	T触发器的T输入
.ld	并行置数输入	.com	组合信号反馈
.oe	输出使能	.le	锁存器的锁存使能输入
.re	异步清零		

7. 数 组

常常将相同性质的变量、信号用一个数组(向量)来表示，由一个符号名来代表数组名。例如：用标识符Out1和A1来表示下列数组：

```
Out1 =[ b0,b1,b2,b3,b4,b5,b6,b7];
A1=[ a2, a1, a0];
```

也可用[..]来表示：

```
Out1 =[b0.. b7];A1=[a2..a0];
```

还可以用数组名后跟元素位置的号码来表示数组的部分元素和一个元素，数组最低位的元素(最左边)的位置号码为0。如Out1[5..2]表示数组Out1中的b2,b3,b4,b5元素，A1[2]表示a0。

数组之间的运算按位(元素)进行运算，但数组必须有相同多的元素。如a=[1,0,1]&[1,1,0]，其结果a=[1,0,0]。

当数值给数组赋值时，系统是先把数值转换为二进制数，如果二进制数的有效位数大于数组元素的个数，那么左边多余的二进制位数将被删除；如果二进制数的有效位数小于数组元素的个数，那么二进制数左边用零补齐(与数组元素相同)。如[a,b]='B1100101与[a,b]='B01等效。[a,b,c]=1等同于a=0,b=0,c=1。

9.3.2 ABEL-HDL的基本结构

一个ABEL_HDL源文件由若干段组成，一般分为5段：标题段、定义段、逻辑描述段、测试向量段和结束段。

1. 标题段

标题段一般包括模块语句和标题语句。

(1) 模块语句

其格式为

MODULE　模块名

模块语句是源文件必须的。它标志着模块开头，必须与结束段的结束语句 end 相呼应。模块名是设计者自定义的名称。放在源文件开头处。

(2) 标题语句(可选)

其格式为

TITLE '……'

标题语句用来对模块加以说明。如电路的内容、用途、作者姓名、时间等。在系统编译时不会处理该语句。

2. 定义段

定义段包括器件定义、信号定义、常量定义、数组定义以及库、宏定义等。其任务是在逻辑描述文前，将本模块所使用的信号名称及其属性和所用的器件、常量、数组、宏等提供给处理程序，紧接标题段后面。其格式为

DECLARATIONS

器件定义；

信号定义；

节点定义；

……

(1) 器件定义(可省略)

其格式为

Device　器件型号；

器件型号为实际器件型号，一般可在 ISP Synario 系统界面的菜单下选择器件型号，因此器件定义可省略。

(2) 信号定义

信号定义语句包括引脚定义和属性定义，引脚定义的关键字为 pin。其格式为

[!]信号名，...[!]信号名 **PIN**　[引脚号，...引脚号]　**ISTYPE**'属性'；

如果不预先设定信号的引脚号，则引脚号可以不填。若在某个信号名之前加有"!"即"非"号，则表示该引脚为低电平有效。

属性是指输出信号的性质，如是组合逻辑输出则用关键字 com 表示；如是寄存器输出则用关键字 reg 表示。信号的属性还有其他定义，如 reg_d(D 触发器)、reg_jk(JK 触发器)。例如：

```
clk              pin ;
out1, out2, out3 pin istype'com';
FF1, FF2, FF3    pin istype'reg';
```

(3) 节点定义

节点定义语句与引脚定义语句结构类似，节点定义语句说明器件内部节点的标识符。节

点是隐埋在器件内部的输入/输出端，对形成中间信号很有用。其格式为

[!]节点名，...[!]节点名 PIN ISTYPE'属性'；

(4) 常量定义

常量定义就是用符号名来代表常数值。用等号"="表示，指定符号常数。其格式为

标识符，...标识符=常数，...常数；

3. 逻辑描述段

在逻辑描述段中可用一个或多个逻辑表达方式来描述逻辑功能，表达方式包括逻辑方程、真值表、状态图。每种表达方式编写的源程序(称为块)有相应的关键字作为块首。

(1) 逻辑方程(块)

逻辑方程每行长度不超过 150 个字符，每句用分号(;)结束。以关键字 Equations 为块首，表示下面为逻辑方程语句。例如：

```
MODULE HD3
  "输入信号
    CLK,JMP,D11..D0      PIN;
  "Outputs
    Q11..Q0              PIN ISTYPE 'REG_D';
  "Nodes
    S                    NODE ISTYPE 'REG_D';
   D=[D11..D0];
   Q=[Q11..Q0];
EQUATIONS                    "逻辑方程"
   S.CLK=CLK;
   S.D=JMP;
   Q.CLK=S.Q;
   Q.D=D;
END
```

在本例中输入信号为 CLK,JMP,D11..D0。输出信号为 Q11..Q0(D 触发器输出)。以一双引号(")开始，以另一双引号(")或行结束表示为注释内容。

(2) 真值表

真值表包括描述组合逻辑电路的真值表，也包括描述时序逻辑电路的状态转换真值表。其格式为

```
Truth_Table
    (输入标识符，...标识符 ->输出标识符，...标识符)    "表头需用圆括号"
        常量，... 常量    ->常量，... 常量；
                    ......
```

其中输入/输出关系为组合型用"->"表示；为寄存型用":>"表示。

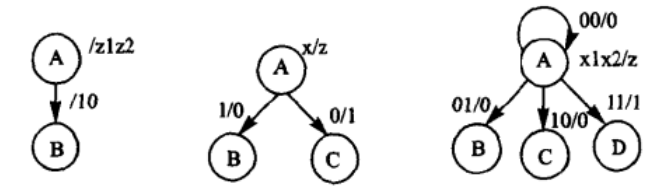
(3) 状态图

在逻辑电路中，状态图用来描述时序逻辑电路的状态转移规律，表现每个状态(现在状态)在一定输入条件下将向什么状态(下一状态)转移，以及伴随状态转移所发生的即刻输出值的变化。其格式为

```
STATE_DIAGRAM 状态名[状态输出]
    STATE 状态表达式:逻辑方程(输出);
                    ...
                    逻辑方程;
                    转移语句;
```

状态图的块首以关键字 STATE_DIAGRAM 开头,后跟状态名(总的状态名)。以关键字 STATE 开始,描述一个当前状态怎样转移到下一状态。状态表达式是表示具体的一个状态(当前状态),冒号后面的逻辑方程是在当前状态下的即刻输出方程,由于状态图中的状态可用表达式、文字、数字等多种方式表示,因此需要预先在源程序定义段中说明(可用数组表示状态位)。

转移语句用来说明状态的转移情况,是状态图描述的关键。转移语句有多种形式,应根据状态转移方式选择合适的转移语句来表示,通常有以下几种,如图 9.3.1 所示。



(a) 无条件转移　(b) 简单分支　(c) 复杂分支

图 9.3.1　状态转移三种方式

① 无条件转移语句(goto 语句)。其格式为

```
GOTO 状态表达式;
```

这里状态表达式为下一状态表达式。以图 9.3.1(a)为例,转移语句为 goto B ;。

② if - then - else 语句。其格式为

```
if  表达式  then    状态表达式
            else    状态表达式;
```

先对 if 后面表达式判断是否为真,若为真则进入 then 后面状态表达式(下一状态),否则进入 else 后面状态表达式(另一状态)。以图 9.3.1(b)为例,转移语句为

```
if x then B else C ;
```

或者

```
if x==1 then B else C ;
```

③ Case 语句。图 9.3.1(c)中有三个或三个以上的分支,每个分支的条件之间总是互斥的,遇到这种复杂分支,宜采用 case 语句。其格式为

```
case 表达式:状态表达式;
     表达式:状态表达式;
          ......
endcase
```

对于图 9.3.1(c)状态图有以下转移语句:

```
state A;
    case !x1&!x2 :A;
```

```
        !x1&x2  :B;
         x1&!x2 :C;
         x1&x2  :D;
endcase
```

④ 状态图中输出信号。时序电路中的输出信号与电路形式有关。摩尔型电路的输出仅取决于电路状态，每个状态有一个确定输出；而米里型电路的输出不仅与状态有关，还与输入有关。

对于摩尔型电路，只要按状态图格式在转移语句前写上输出逻辑方程即可；对于米里型电路的输出，应利用 with - endwith 语句，其格式为

```
转移语句  状态表达式  with  输出方程;
                            输出方程;
                      endwith
```

其中转移语句是指 goto，if - then - else，case 语句，状态表达式是下一状态，with 后面是即刻输出方程。

对于图 9.3.1(a)有

```
state A :goto B with z1=1;
                     z2=0;
          endwith
```

对于图 9.3.1(b)有

```
state A :if x then B with z=0;
                  endwith
        else C with z=1;
                  endwith
```

对于图 9.3.1(c)有

```
state A:
    case !x1&!x2 : A with z=0;
                   endwith
         !x1&x2  : B with z=0;
                   endwith
          x1&!x2 : C with z=0;
                   endwith
          x1&x2  : D with z=1;
                   endwith
                   endcase
```

上面各例中的输出都是在当前状态下的输出信号，而下一状态相对是延迟的。有时希望电路的输出是状态转移以后的输出，即与下一状态同步输出。这时 with 后面的输出方程使用寄存器赋值符(:=)。如：

```
state A :if x then B  with z:=0;
              endwith
        else C  with z:=1;
              endwith
```

4. 测试向量段

测试向量段是可选内容，用来检查逻辑段描述的电路设计是否能实现预期的设计功能。引导测试向量段的关键字为 TEST - VECTORS 。测试向量段可以单独作为一个文件来编写。

测试向量的格式类真值表，也有一个用圆括号括出的表头，表头和向量表各行的左边是输入向量，右边是输出向量(一般用数组表示)，中间由符号－＞连接。

5. 结束段

其格式为

END

它标志一个模块的结束。

ABEL - HDL 语言是较为通用的语言，内容相对较复杂，详细内容请参考 ABEL - HDL 手册。

6. ABEL - HDL 语言设计举例

例 9.3.1　设计一个十位二进制加法计数器，并且具有保持功能。其源文件如下：

```
MODULE COUNTER10
TITLE 'Count until HOLD active'
    C,P,X=.c.,.p.,.x.;                    "Constants
    !CLK,!RST,HOLD     pin2,3,4;          "Inputs
"Outputs
    Q9,Q8,Q7,Q6,Q5     pin 29,28,27,26,25 istype 'reg_d';
    Q4,Q3,Q2,Q1,Q0     pin 19,18,17,16,15 istype 'reg_d';
    COUNT=[Q9,Q8,Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0]; "Set declarations
Equations
    COUNT :=(COUNT+1)&!HOLD       "Count up if not HOLD
            #(COUNT)&HOLD;        "Hold count
    COUNT.AR=RST;
    COUNT.C=CLK;
Test_Vectors ([CLK, RST,HOLD,!COUNT] -> COUNT)
          [1 , 0 , 0 , X ] -> X ;
          [1 , 1 , 0 , X ] -> 0 ;
          [1 , 0 , 0 , X ] -> 0 ;
          [C , 0 , 0 , X ] -> 1 ;
          [C , 0 , 0 , X ] -> 2 ;
          [C , 0 , 0 , X ] -> 3 ;
          [C , 0 , 0 , X ] -> 4 ;
          [C , 0 , 0 , X ] -> 5 ;
          [C , 0 , 0 , X ] -> 6 ;
          [C , 0 , 0 , X ] -> 7 ;
          [C , 0 , 0 , X ] -> 8 ;
          [C , 0 , 0 , X ] -> 9 ;
          [C , 0 , 0 , X ] -> 10 ;
```

```
        [C , 0 , 0 , X ] -> 11 ;
        [C , 0 , 0 , X ] -> 12 ;
        [C , 0 , 0 , X ] -> 13 ;
        [C , 0 , 0 , X ] -> 14 ;
        [P , 0 , 0 , 1019] -> X ;
        [0 , 0 , 0 , X ] -> 1019 ;
        [C , 0 , 0 , X ] -> 1020 ;
        [C , 0 , 0 , X ] -> 1021 ;
        [C , 0 , 0 , X ] -> 1022 ;
        [C , 0 , 0 , X ] -> 1023 ;
        [C , 0 , 1 , X ] -> 1023 ;
        [C , 0 , 1 , X ] -> 1023 ;
        [C , 0 , 1 , X ] -> 1023 ;
        [0 , 1 , 0 , X ] -> 0 ;
END
```

例 9.3.2 设计 BCD 码七段显示译码器的源文件如下：

```
MODULE BCD_LED
TITLE '8 SEGMENT LED DRIVER'
"Inputs
    d0,d1,d2,d3          PIN;
"Outputs
    a,b,c,d,e,f,g        PIN ISTYPE 'COM';
TRUTH_TABLE ([d3,d2,d1,d0]-> [a,b,c,d,e,f,g])
        'H0    -> [1,1,1,1,1,1,0];
        'H1    -> [0,1,1,0,0,0,0];
        'H2    -> [1,1,0,1,1,0,1];
        'H3    -> [1,1,1,1,0,0,1];
        'H4    -> [0,1,1,0,0,1,1];
        'H5    -> [1,0,1,1,0,1,1];
        'H6    -> [1,0,1,1,1,1,1];
        'H7    -> [1,1,1,0,0,0,0];
        'H8    -> [1,1,1,1,1,1,1];
        'H9    -> [1,1,1,1,0,1,1];
        'HA    -> [1,1,1,0,1,1,1];
        'HB    -> [0,0,1,1,1,1,1];
        'HC    -> [1,0,0,1,1,1,0];
        'HD    -> [0,1,1,1,1,0,1];
        'HE    -> [1,0,0,1,1,1,1];
        'HF    -> [1,0,0,0,1,1,1];
END
```

第十章　常用数字电路的设计实例

现在可利用前面章节介绍的 MAX+PLUSⅡ和 ISP Synario 开发软件工具，进行数字电路的设计。设计输入方式可采用图形(原理图)和文本(AHDL，ABEL 语言)输入方式，在顶层文件中只能采用图形输入方式。在许多开发软件工具中提供了大量常用的数字电路部件，以便设计中调用。下面介绍常用数字单元电路的设计。

10.1　组合逻辑电路

1. 编码器和译码器

例 10.1.1　设计一个 8 线-3 线优先编码器。其功能表如表 10.1.1 所示。

表 10.1.1　8 线-3 线优先编码器功能表

输　入								输　出		
D7	D6	D5	D4	D3	D2	D1	D0	Y2	Y1	Y0
1	×	×	×	×	×	×	×	1	1	1
0	1	×	×	×	×	×	×	1	1	0
0	0	1	×	×	×	×	×	1	0	1
0	0	0	1	×	×	×	×	1	0	0
0	0	0	0	1	×	×	×	0	1	1
0	0	0	0	0	1	×	×	0	1	0
0	0	0	0	0	0	1	×	0	0	1
0	0	0	0	0	0	0	1	0	0	0

设计步骤如下：

① 建立原理图(顶层文件)，如图 10.1.1 所示。

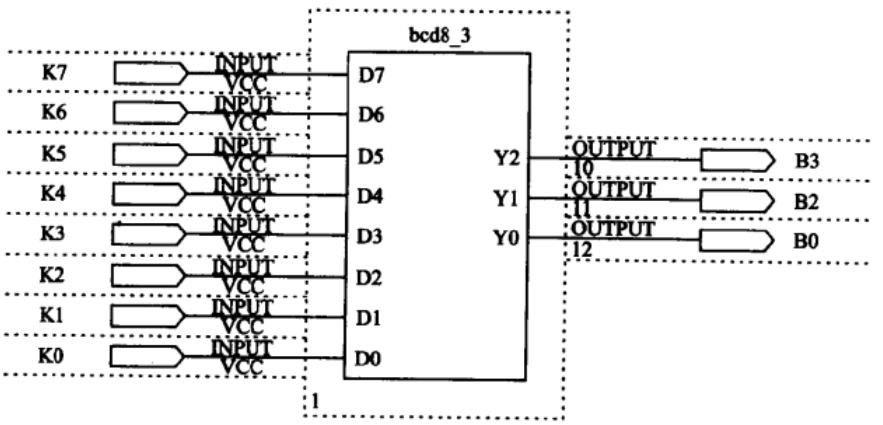


图 10.1.1　优先编码器原理图

② 用 AHDL 语言描述优先编码器 bcd8_3 的功能(底层文件),其源文件为

```
SUBDESIGN bcd8_3
(
    D7,D6,D5,D4,D3,D2,D1,D0   : INPUT ;
    Y[2..0]                   : OUTPUT ;
)
BEGIN
  IF D7     THEN
     Y[]=7;
  ELSIF D6 THEN
     Y[]=6;
  ELSIF D5 THEN
     Y[]=5;
  ELSIF D4 THEN
     Y[]=4;
  ELSIF D3 THEN
     Y[]=3;
  ELSIF D2 THEN
     Y[]=2;
  ELSIF D1 THEN
     Y[]=1;
  ELSE
     Y[]=0;
  END IF;
END;
```

③ 指定器件(EPM7128S),分配引脚,进行编译,然后下载数据,进行器件编程。

例 10.1.2 设计 3 线-8 线译码器。其设计方法与编码器相似,首先建立顶层文件(原理图),如图 10.1.2 所示;然后用 AHDL 语言编写 abcd3_8 模块的源文件即可。

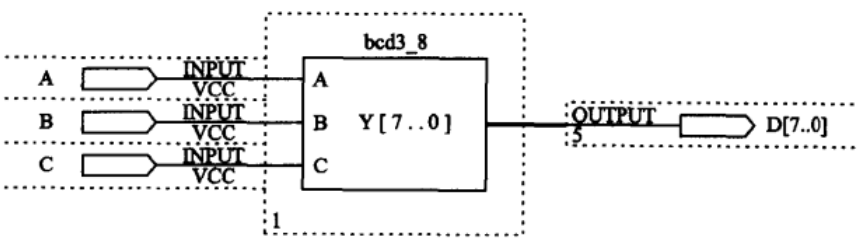


图 10.1.2 3 线-8 线译码器原理图

bcd3_8 模块的源文件描述为

```
SUBDESIGN abcd3_8
(  a,b,c   : INPUT;
  Y[7..0] : OUTPUT;
  )
BEGIN
```

```
CASE (a,b,c) IS
    WHEN 0 => Y[7..0]=1;
    WHEN 1 => Y[7..0]=2;
    WHEN 2 => Y[7..0]=4;
    WHEN 3 => Y[7..0]=8;
    WHEN 4 => Y[7..0]=16;
    WHEN 5 => Y[7..0]=32;
    WHEN 6 => Y[7..0]=64;
    WHEN OTHERS=> Y[7..0]=128;
 END CASE;
END;
```

2. 码变换电路

例 10.1.3　设计 4 位二进制数/8421BCD 码的变换电路。要求将 4 位二进制数(0～F)表示为十进制数(0～15)，因此需要输出变量为 5 位(D10，D03，D02，D01，D00)。其顶层文件(原理图)如图 10.1.3 所示。

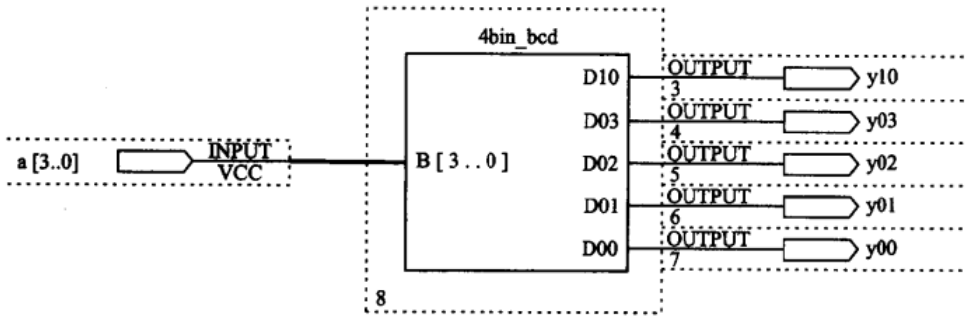


图 10.1.3　4 位二进制数/8421BCD 码的变换电路

4bin_bcd 模块的源文件描述为

```
    SUBDESIGN 4bin_bcd
(
    b[3..0]                 : INPUT;
    d10,d03,d02,d01,d00     : OUTPUT;
 )
 BEGIN
    TABLE                                   --真值表描述
       b[3..0] => d10, d0[3..0];
       B"0000" =>  0 , B"0000" ;
       B"0001" =>  0 , B"0001" ;
       B"0010" =>  0 , B"0010" ;
       B"0011" =>  0 , B"0011" ;
       B"0100" =>  0 , B"0100" ;
       B"0101" =>  0 , B"0101" ;
       B"0110" =>  0 , B"0110" ;
       B"0111" =>  0 , B"0111" ;
```

```
        B"1000" =>  0 , B"1000" ;
        B"1001" =>  0 , B"1001" ;
        B"1010" =>  1 , B"0000" ;
        B"1011" =>  1 , B"0001" ;
        B"1100" =>  1 , B"0010" ;
        B"1101" =>  1 , B"0011" ;
        B"1110" =>  1 , B"0100" ;
        B"1111" =>  1 , B"0101" ;
    END TABLE;
END;
```

3. 数码显示电路

例 10.1.4 设计一个七段数码显示译码器。首先建立原理图，如图 10.1.4 所示。

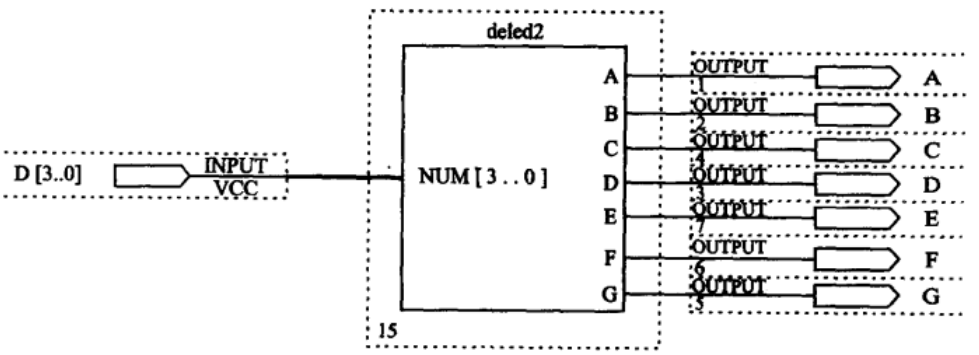


图 10.1.4 七段显示译码器

Deled1 模块的源文件描述如下：

```
    SUBDESIGN deled1
(
  num[3..0]:INPUT;
  a,b,c,d,e,f,g: OUTPUT;              --七段输出
      )
BEGIN
  TABLE
    num[3..0]=> a,b,c,d,e,f,g;
    H"0"      => 1,1,1,1,1,1,0;      --显示 0
    H"1"      => 0,1,1,0,0,0,0;      --显示 1
    H"2"      => 1,1,0,1,1,0,1;      --显示 2
    H"3"      => 1,1,1,1,0,0,1;      --显示 3
    H"4"      => 0,1,1,0,0,1,1;      --显示 4
    H"5"      => 1,0,1,1,0,1,1;      --显示 5
    H"6"      => 1,0,1,1,1,1,1;      --显示 6
    H"7"      => 1,1,1,0,0,0,0;      --显示 7
    H"8"      => 1,1,1,1,1,1,1;      --显示 8
    H"9"      => 1,1,1,1,0,1,1;      --显示 9
```

```
        H"A"    => 1,1,1,0,1,1,1;    -- 显示 A
        H"B"    => 0,0,1,1,1,1,1;    -- 显示 B
        H"C"    => 1,0,0,1,1,1,0;    -- 显示 C
        H"D"    => 0,1,1,1,1,0,1;    -- 显示 D
        H"E"    => 1,0,0,1,1,1,1;    -- 显示 E
        H"F"    => 1,0,0,0,1,1,1;    -- 显示 F
    END TABLE;
END;
```

4. 简单组合电路

例 10.1.5　设计四个开关控制一盏灯的逻辑电路，要求任何一开关能控制灯亮和灭。采用 AHDL 语言描述的源文件为

```
SUBDESIGN kanguan
(
    K2,K1,K0    : INPUT ;
    OUT         : OUTPUT ;
)

BEGIN
    TABLE
        K2,K1,K0 => OUT ;
          B"000"  =>  0 ;
          B"001"  =>  1 ;
          B"010"  =>  1 ;
          B"100"  =>  1 ;
          B"101"  =>  0 ;
          B"110"  =>  0 ;
          B"011"  =>  0 ;
          B"111"  =>  1 ;
        END TABLE;
END;
```

其电路原理图如图 10.1.5 所示。

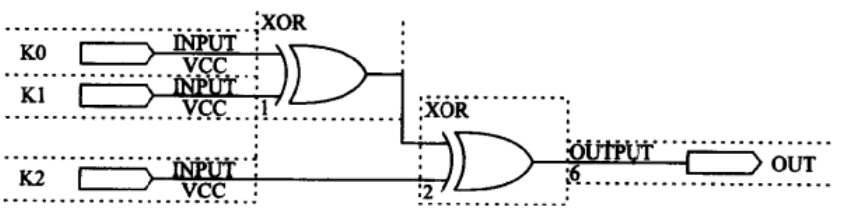


图 10.1.5　开关控制逻辑电路

例 10.1.6　设计函数发生器，实现下列函数：

$$\begin{cases} F1 = A\overline{B} + \overline{B}C + AC \\ F2 = \overline{A}\,\overline{B} + B\overline{C} + ABC \\ F3 = \overline{A}C + BC + A\overline{C} \end{cases}$$

由于这是一组 3 输入变量的多输出逻辑函数，因此可以采用 3 线-8 线译码器设计。这里直接调用元件函数模块(元件库)中的 3 线-8 线译码器 74138 和“与非”门，通过电路原理图来实现。首先将多输出逻辑函数写成最小项表达式

$$F_1 = A\overline{B} + \overline{B}C + AC = m_1 + m_4 + m_5 + m_7 = \overline{\overline{Y1}\,\overline{Y4}\,\overline{Y5}\,\overline{Y7}}$$

$$F_2 = \overline{A}\,\overline{B} + B\overline{C} + ABC = m_0 + m_1 + m_2 + m_6 + m_7 = \overline{\overline{Y0}\,\overline{Y1}\,\overline{Y2}\,\overline{Y6}\,\overline{Y7}}$$

$$F_3 = \overline{A}C + BC + A\overline{C} = m_1 + m_3 + m_4 + m_6 + m_7 = \overline{\overline{Y1}\,\overline{Y3}\,\overline{Y4}\,\overline{Y6}\,\overline{Y7}}$$

实现 F1,F2,F3 函数的逻辑电路如图 10.1.6 所示。

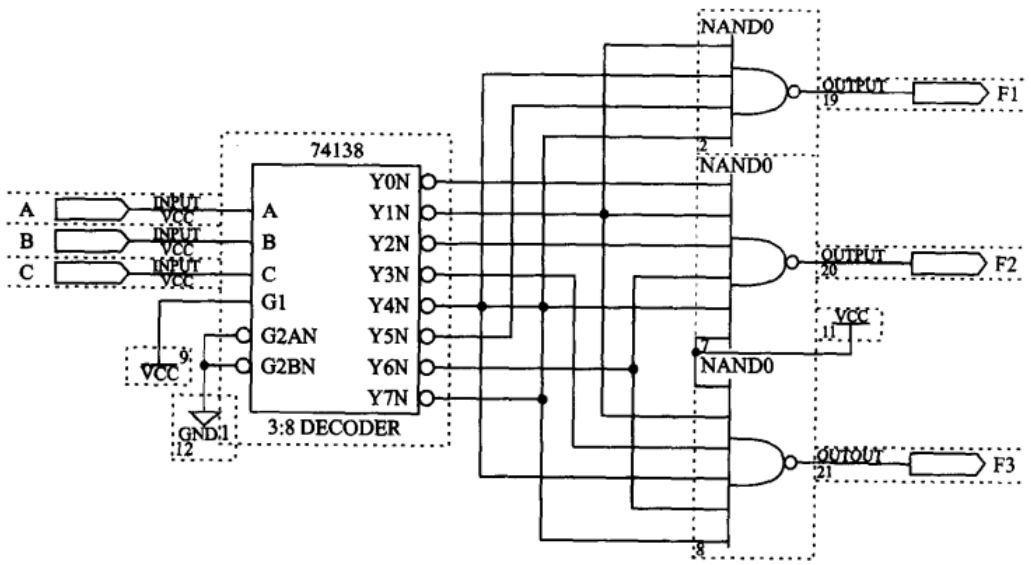


图 10.1.6 实现 F1,F2,F3 函数的逻辑电路

10.2 寄存器和计数器

在时序逻辑电路图中，通过触发器的连接，实现寄存器和计数器。MAX+PLUSⅡ软件工具中的每种触发器有两类，例如 D 触发器分为 DFF 和 DFFE，如图 10.2.1 所示。

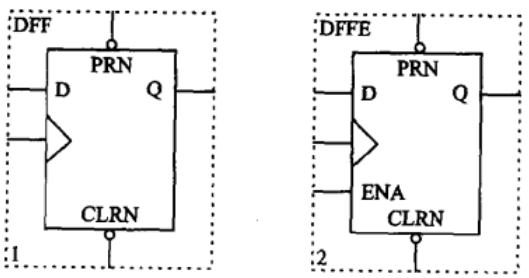


图 10.2.1 两类 D 触发器

DFF 是通常的 D 触发器。它有时钟端、数据输入端(D)、输出端(Q)、异步置位端(PRN 低电平有效)和异步清零端(CLRN 低电平有效)。DFFE 触发器增加了一个时钟使能端 ENA。当 ENA=1 时,DFFE 功能与 DFF 相同;当 ENA=0 时,即使有时钟作用,触发器仍维持原来的状态。对于寄存器和计数器的设计,可采用原理图和文本输入方式描述,这里使用文本输入方式(AHDL 语言)来设计。

例 10.2.1　一个 8 位寄存器的 AHDL 源文件描述如下:

```
SUBDESIGN 8reg
(
        clk,load,d[7..0]     : INPUT ;
        q[7..0]              : OUTPUT;
)
VARIABLE
        ff[7..0]:DFFE;
BEGIN
        ff[].clk=clk;
        ff[].ena=load;
        ff[].d=d[];
        q[]=ff[].q;
END;
```

其中 d[7..0]为并行数据输入信号,clk 为时钟输入信号,load 为置数控制输入信号,q[7..0]为寄存器输出信号。并建立了该 8 位寄存器的逻辑符号,如图 10.2.2 所示。

例 10.2.2　10 bit 移位寄存器的 AHDL 源文件描述如下:

```
    SUBDESIGN shift10
(
    in,clock          : INPUT ;
    out, q[9..0]      : OUTPUT;
)
VARIABLE
    ff[9..0]          : DFF;
BEGIN
   ff[].clk=clock;
   q[]=ff[].q;
   ff[9..1].d=ff[8..0].q;    --触发器向下移位
   ff0.d=in;
   out=ff9.q;
END;
```

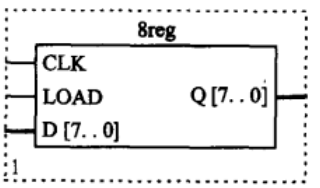


图 10.2.2　8 位寄存器逻辑符号

其中 in 为串行数据输入,out 为串行数据输出,q[9..0]为状态的 10 位并行输出。

例 10.2.3　设计一个模可变的 6 位二进制加法计数器,可通过计数模式 M[1..0]选择输入,实现最多为 4 种不同模式的计数方式,假如可构成六进制、十二进制、二十四进制和六十进制共 4 种计数模式。此计数器的 AHDL 源文件描述如下:

```
    SUBDESIGN 6bin_cnt                          -- 可变模计数器
(
    clk, clr, m[1..0]          : INPUT;
    q[5..0], co                : OUTPUT;        -- co 进位输出
)
VARIABLE
    count[5..0]                : DFF;
BEGIN
    count[].clk = clk;
    count[].clrn =! co&! clr;
 CASE m[1..0] IS
     WHEN 0 =>
       IF count[].q<6 THEN                      -- m[1..0]=00 时,六进制计数
          count[].d = count[].q + 1;
          co=GND;
       ELSE
          co=VCC;
       END IF;
     WHEN 1=>
       IF count[].q<12 THEN                     -- m[1..0]=01 时,十二进制计数
          count[].d = count[].q + 1;
          co=GND;
       ELSE
          co=VCC;
       END IF;
    WHEN 2=>
       IF count[].q<24 THEN                     -- m[1..0]=10 时, 二十四进制计数
          count[].d = count[].q + 1;
          co=GND;
       ELSE
          co=VCC;
       END IF;
    WHEN 3=>
       IF count[].q<60 THEN                     -- m[1..0]=11 时, 六十进制计数
          count[].d = count[].q + 1;
          co=GND;
       ELSE
          co=VCC;
       END IF;
     WHEN OTHERS =>
        count[].d = 0 ;
END CASE;
    q[] = count[];
```

END;

其中 clk 为时钟输入信号，clr 为异步清零输入（高电平有效），q[5..0] 为状态输出，co 为进位输出。

例 10.2.4　设计一个模为 100 的十进制加法计数器（BCD 码输出），需要用两位数码管同时顺序显示十进制 00～99。

首先用两片十进制计数器 74160 组成一百进制计数器，输出高 4 位 BCD 码（DA7，DA6，DA5，DA4）和低 4 位 BCD 码（DA3，DA2，DA1，DA0）。然后采用扫描显出方式，动态显示两位数码管，因此使用一片七段显示译码器，通过数码管的片选端（公共端为阴极），轮流显示两位数码管。该计数器的顶层文件（原理图）描述如图 10.2.3 所示。

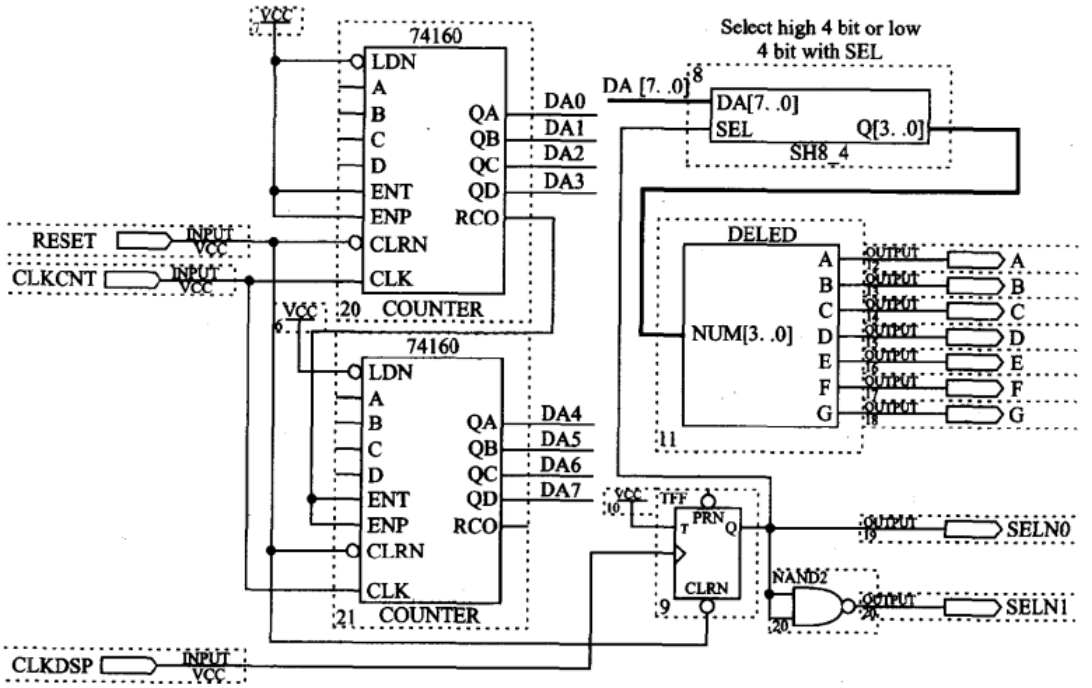


图 10.2.3　一百进制计数器原理图

其中输入时钟信号 CLKCNT 为计数时钟（频率＜4 Hz），CLKDSP 为扫描时钟（频率＞100 Hz）。输出信号 SELN0 和 SELN1 为扫描地址，分别控制各对应数码管的显示。DELED 模块为七段显示译码器。模块 SH8_4 完成高 4 位 BCD 码和低 4 位 BCD 码数据的切换，当 SEL=0 时，Q[3..0]=DA[3..0]（低 4 位 BCD 码）；当 SEL=1 时，Q[3..0]=DA[7..4]（高 4 位 BCD 码）。SH8_4 模块的 AHDL 源文件描述如下：

```
SUBDESIGN sh8_4
(
    sel,da[7..0]: input;
    q[3..0]     : output;
)
BEGIN
    IF !sel THEN        -- SEL=0 时，低 4 位 BCD 码输出
```

```
        q[]=da[3..0];
      ELSE
        q[]=da[7..4];      -- SEL=1 时,高 4 位 BCD 码输出
      END IF;
END;
```

10.3 有限状态机设计

状态机就是一组触发器的输出状态随着时钟和输入信号(一个或多个)按照一定规律变化的过程,常用状态转移图(或表)来表示。

例如,一个双向步进电机控制电路的状态转移图,如图 10.3.1 所示。该控制电路的输入信号有 3 个:时钟 clk、复位 reset、方向控制 RL。输出信号为 Y[3..0],用来控制电机的动作,每个状态对应一组不同输出信号。当方向控制信号 RL=1 时,状态机随时钟按 $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_0$ 正向循环。当 RL=0 时,状态机随时钟按 $s_0 \rightarrow s_3 \rightarrow s_2 \rightarrow s_1 \rightarrow s_0$ 反向循环。

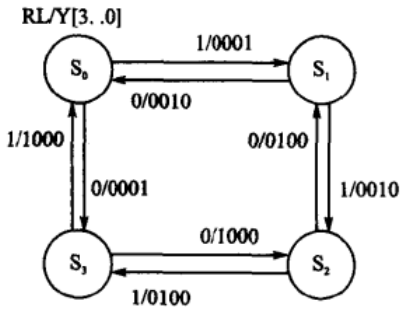


图 10.3.1 状态转移图

以下是双向步进电机控制电路的 AHDL 源文件描述:

```
        SUBDESIGN stepper
(
    clk,reset,RL     : INPUT;
    Y[3..0]          : OUTPUT;
)
VARIABLE
   ss: MACHINE  WITH STATES (s0, s1,
s2,s3);
BEGIN
    ss.clk=clk;
    ss.reset=reset;
  TABLE
    ss, RL   => ss,    Y[3..0];
    s0, 0    => s3,    B"0001";
    s0, 1    => s1,    B"0001";
    s1, 0    => s0,    B"0010";
    s1, 1    => s2,    B"0010";
    s2, 0    => s1,    B"0100";
    s2, 1    => s3,    B"0100";
    s3, 0    => s2,    B"1000";
    s3, 1    => s0,    B"1000";
  END TABLE;
END;
```

10.4　综合电路设计

1. 汽车尾灯控制电路

该电路控制汽车尾部左右两侧各 3 个指示灯，要求如下：

① 汽车正常运行时两侧指示灯全灭；当刹车时，尾部两侧指示灯全亮。

② 右转弯时，右侧 3 个指示灯按 000→100→010→001→000 循环顺序点亮，左侧灯全灭；左转弯时，左侧 3 个指示灯也按同样循环顺序点亮，右侧灯全灭。

③ 在转弯刹车时，向转弯这侧的三个尾部灯按同样循环顺序点亮，另一侧的灯全亮。

首先输入信号分为三个：R 为右转弯控制，L 为左转弯控制，C 为刹车输入。其电路系统的原理图如图 10.4.1 所示(即顶层文件)，分为转弯控制模块和循环灯显示模块。转弯控制模块的控制输出有：左侧灯按循环顺序点亮控制 ML 和全亮控制 ZL；右侧灯按循环顺序点亮控制 MR 和全亮控制 ZR 。循环灯显示模块是由两个元件库函数 74195 分别组成左、右两侧尾灯显示电路。

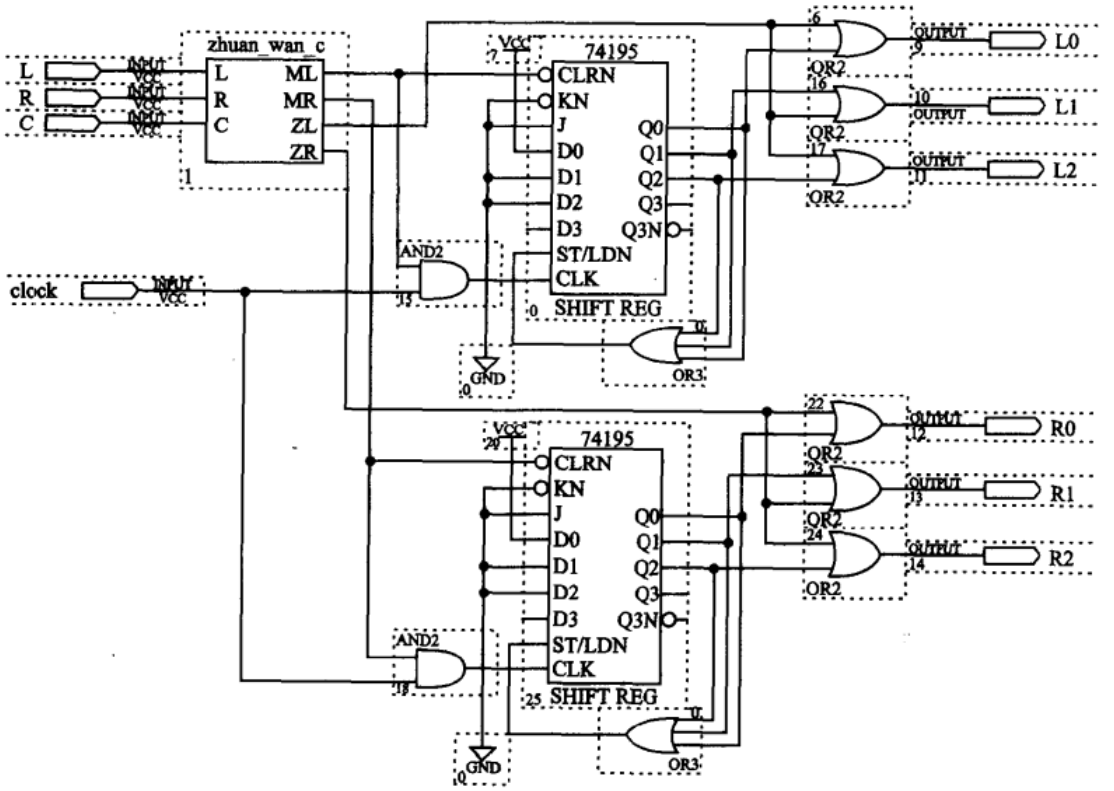


图 10.4.1　汽车尾灯显示电路

其中转弯控制模块的源文件(AHDL)描述如下：

```
SUBDESIGN zhuan_wan_c
(
        L,R,C       : INPUT ;
```

```
        ML,MR,ZL,ZR : OUTPUT;
    )
BEGIN
    TABLE
        L,R,C => ML, MR, ZL, ZR ;
        0,0,0 =>  0,0,0,0;
        0,0,1 =>  0,0,1,1;
        0,1,0 =>  0,1,0,0;
        0,1,1 =>  0,1,1,0;
        1,0,0 =>  1,0,0,0;
        1,0,1 =>  1,0,0,1;
        1,1,0 =>  x,x,x,x;
        1,1,1 =>  x,x,x,x;
    END TABLE;
END;
```

2. 竞赛抢答器电路

该系统设计要求如下：

① 有若干队参加竞赛(设7队)，每队对应一个抢答按钮，还有按钮给主持人用来清零。

② 抢答器具有数据锁存功能，对输入信号有很强的分辨能力，只显示先抢答队的号数(用LED数码管显示)，并且发出声响，直到主持人清零为止。

根据设计要求，进行方案确定。该系统电路分为两大模块，分为控制模块和显示模块。图10.4.2给出了竞赛抢答器电路的原理图。控制模块(qan_da_c)需完成输入信号的分辨、数据锁存的功能，采用硬件描述语言进行该模块描述。显示模块主要由编码器(bcd8_3)和七段显示译码器(deled1)组成。

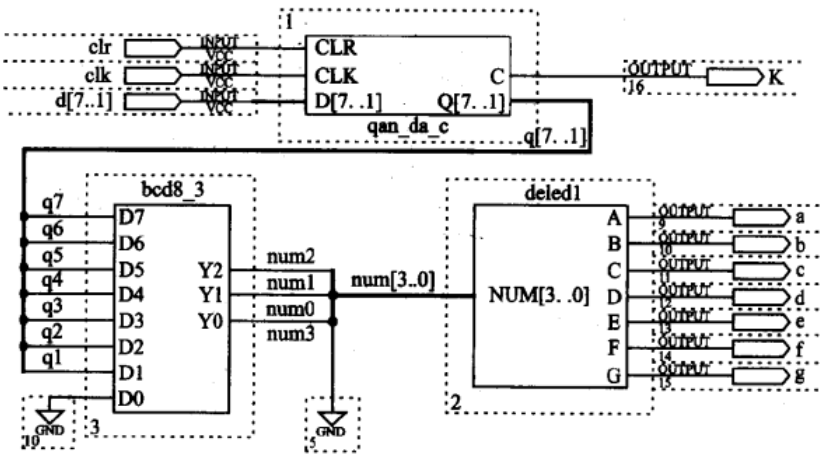


图 10.4.2 竞赛抢答器电路的原理图

控制模块(qan_da_c)的AHDL源文件描述如下：

```
    SUBDESIGN qan_da_c
(
    d[7..1], clk, clr    : INPUT ;
```

```
    q[7..1], c          : OUTPUT;
)
VARIABLE
    ff[7..1]            : DFFE;

BEGIN
    ff[].clk=clk;
    ff[].clrn=!clr;
    ff[].d=d[];
    q[]=ff[].q;
    c=ff1 # ff2 # ff3 # ff4 # ff5 # ff6 # ff7;
    ff[].ena=!c;
END;
```

其中 d[7..1]是抢答按钮输入端(高电平有效),clk 为时钟,clr 为异步清零端(高电平有效),输出信号 c 为有信号输入时,产生逻辑"1";无信号输入或清零后,输出信号 c 为逻辑"0"。编码器(bcd8_3)和七段显示译码器(deled1)已在前面介绍过。

3. 数字钟

数字钟设计要求(数字钟功能)如下:

① 具有时、分、秒计数显示功能,以小时循环计时;

② 具有清零,调节小时、分钟功能;

③ 具有整点报时功能,并且伴随 LED 灯花样显示。

该系统的原理图(顶层文件)如图 10.4.3 所示,共有六个模块。其中模块 second,minute 和 hour 分别为秒、分、时的计数模块;模块 alert 为彩灯和扬声器编码模块;sel_time 为扫描数据(BCD 码)的切换模块;deled 为七段显示译码器。

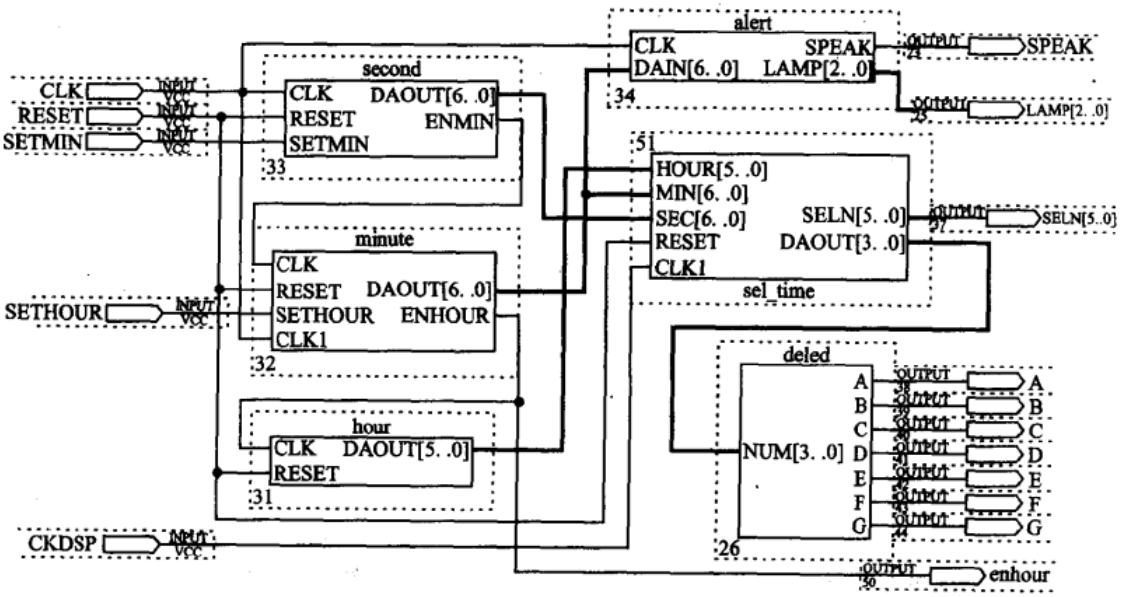


图 10.4.3　数字钟的原理图(顶层文件)

其中输入端 CLK 为计数时钟(1 Hz),CKDSP 为扫描时钟(>100 Hz);RESET,SETHOUR 和 SETMIN 分别为清零、调时和调分输入信号。输出信号有数码管的驱动信号 A～G 和片选信号 SETN[5..0](轮流显示六位数码管);在整点报时的输出信号(持续 1 min)为扬声器驱动信号 SPEAK(0.5 Hz)、花样 LED 灯显示信号 LAMP[2..0]和整点脉冲信号 enhour。以下是用 AHDL 语言对这些模块的功能描述。

(1) 模块 second(秒计数)的源文件描述

```
SUBDESIGN second1
(
  clk,reset,setmin     :INPUT;          --计数时钟 clk 为 1 Hz
  daout[6..0],enmin    :OUTPUT;         --enmin 为秒计数进位输出
)
VARIABLE
  count[6..0],s:    DFF;

BEGIN
  count[].clk=clk;
  count[].clrn=!reset;                  --复位端 reset 高电平有效
  daout[]=count[].q;
  s.clk=clk;
  s.clrn=!reset;
  enmin=s.q;
  IF setmin THEN                        --调分计数输入 setmin(高电平有效)
    enmin=clk;
  END IF;

  IF (count[].q<H"59") THEN             --六十进制计数
    IF(count[3..0].q>=9) THEN
      count[].d=count[].q+7;            --二进制数转换为 BCD 码
    ELSE
      count[].d=count[].q+1;
    END IF;
  ELSE
    count[].d=0;
    s.d=VCC;
  END IF;
END;
```

(2) 模块 minute(分计数)的源文件描述

```
SUBDESIGN minute
(
  clk,reset,sethour,clk1     : INPUT ;     --clk 连接秒计数进位输出 enmin
  daout[6..0],  enhour       : OUTPUT;     --enhour 为分计数进位输出
)
VARIABLE
```

```
    count[6..0], s          : DFF;

BEGIN
    count[].clk=clk;
    count[].clrn=!reset;                    -- 复位端 reset 高电平有效
    daout[]=count[].q;
    s.clk=clk;
    s.clrn=!reset;
    enhour=s.q;
  IF sethour THEN                           -- 调时计数输入 sethour(高电平有效)
    enhour=clk1
-- ELSE
--   enhour=s.q;
  END IF;

  IF(count[].q<H"59") THEN                  -- 六十进制计数
     IF (count[3..0].q>=9) THEN
        count[].d=count[].q+7;              -- 二进制数转换为 BCD 码
      ELSE
        count[].d=count[].q+1;
      END IF;
    ELSE
       count[].d=0;
       s.d=VCC;
    END IF;
END;
```

(3) 模块 hour(时计数)的源文件描述

```
    SUBDESIGN hour
(
   clk,  reset          : INPUT;           -- clk 连接分计数进位输出 enhour
   daout[5..0]          : OUTPUT;
)
VARIABLE
   count[5..0]          : DFF;
BEGIN
   count[].clk=clk;
   count[].clrn=!reset;
   daout[]=count[].q;
   IF (count[].q<H"23") THEN                -- 二十四进制计数
      IF (count[3..0].q==9) THEN
        count[].d=count[].q+7;              -- 二进制数转换为 BCD 码
      ELSE
        count[].d=count[].q+1;
      end if;
```

```
    ELSE
        count[].d=0;
    END IF;
END;
```

(4) 模块 alert 的源文件描述

```
SUBDESIGN alert
(
    clk, dain[6..0]          : INPUT;
    speak, lamp[2..0]        : OUTPUT;
)
VARIABLE
    s:    DFF;
    ss:   MACHINE OF BITS (lamp[2..0])
          WITH STATES(
               s0=B"000",          -- 输出信号 lamp[2..0]= 000
               s1=B"001",          -- 输出信号 lamp[2..0]= 001
               s2=B"010",          -- 输出信号 lamp[2..0]= 010
               s3=B"100");         -- 输出信号 lamp[2..0]= 100

BEGIN
      ss.clk=clk;
    IF(dain[]==0) THEN
      s.clk=clk;
      s.d=!s.q;
      speak=s.q;                   -- 输出信号 speak 是时钟 clk 的二分频
     CASE ss IS
      WHEN s0 =>
              ss=s1;
      WHEN s1 =>
              ss=s2;
      WHEN s2 =>
              ss=s3;
      WHEN s3 =>
              ss=s1;
      WHEN OTHERS =>
               ss=s0;
     END CASE;
    ELSE
       ss=s0;
       speak=GND;
    END IF;
END;
```

(5) 模块 sel_time 的源文件描述

```
SUBDESIGN sel_time
```

```
(
    clk1,reset,sec[6..0],min[6..0],hour[5..0]: INPUT;
    daout[3..0],  seln[5..0]                   : OUTPUT;
)
VARIABLE
     count[2..0]                               : DFF;

BEGIN
       count[].clk=clk1;
       count[].clrn=reset;
     IF(count[].q>=5) THEN                     --六进制计数
        count[].d=0;
     ELSE
        count[].d=count[].q+1;
     END IF;
  TABLE
     count[].q   => seln[] ;
         0       => B"111110" ;
         1       => B"111101" ;
         2       => B"111011" ;
         3       => B"110111" ;
         4       => B"101111" ;
         5       => B"011111" ;
         6       => B"xxxxxx" ;
         7       => B"xxxxxx" ;
  END TABLE;
  CASE count[] IS
      WHEN 0     =>
                 daout[]=sec[3..0];        --秒的低4位BCD码(个位)
      WHEN 1     =>
                 daout3=GND;
                 daout[2..0]=sec[6..4];    --秒的高4位BCD码(十位)
      WHEN 2     =>
                 daout[]=min[3..0];        --分的低4位BCD码(个位)
      WHEN 3     =>
                 daout3=GND;
                 daout[2..0]=min[6..4];    --分的高4位BCD码(十位)
      WHEN 4     =>
                 daout[]=hour[3..0];       --小时的低4位BCD码(个位)
      WHEN OTHERS =>
                daout[3..2]=0;
                daout[1..0]=hour[5..4];    --小时的高4位BCD码(十位)
  END CASE;
END;
```

下篇　现代数字系统设计

第十一章　数字系统设计基础

11.1　数字系统设计概述

随着数字集成技术和电子设计自动化(EDA)技术的发展，数字系统设计的理论和方法也随之发生了很大变化和发展，传统的设计方法逐步被基于 EDA 技术的芯片设计方法所替代。设计过程的自动化程度得到很大提高，将过去传统搭积木式的设计方式(自底向上)变成一种自顶向下的设计方法。

数字系统的实现经历了由 SSI，MSI，LSI 到 VLSI 的过程，数字器件经历了由通用集成电路到专用集成电路的变化过程。虽然实现数字系统的器件和方法有多种多样，但基本概念、基本理论仍是非常重要的。

11.1.1　数字系统的基本概念

数字系统是由对信息进行采集、转换、传输、存储、加工处理和利用的一组相互联系、相互作用的部件所组成的一个有机整体。尽管信息具有各种各样的形态和特征，如离散的、连续的、机械运动的速度与位移、商品行情的经济信息、图文信息等。所有这些信息经过变换，转换成数字系统所能接收的数字信息，加以存储和处理。反过来，数字系统加工处理后的信息经过相应的逆变换，成为对被控对象进行有效控制的信号或进行管理和决策的可靠依据。

数字系统与模拟系统相比，具有如下特点。

① 稳定性。数字系统所加工处理的信息是离散的数字量，对用来构成系统的电子元器件要求不高，即能以较低的硬件实现较高的性能。

② 精确性。数字系统中可用增加数据位数或长度来达到数据处理和传输的精确度。

③ 可靠性。数字系统中可采用检错、纠错和编码等信息冗余技术，以及多机并行工作等硬件冗余技术来提高系统的可靠性。

④ 模块化。把系统分成不同功能模块，由相应的功能部件来实现，从而使系统的设计、试制、生产、调试和维护都十分方便。

数字系统一般由若干数字电路和逻辑功能部件组成，并由一个控制部件统一指挥。逻辑部件担负系统的局部任务，完成子系统的功能。把多个子系统构成为大系统时，就必须有一个控制部件来统一协调和管理各子系统的工作，按一定程序指挥整个系统的工作。因此有没有

控制部件是区别数字系统和逻辑功能部件(数字单元电路)的重要标志。凡是有控制部件,并且能按照一定程序进行操作的系统,不论其规模大小,均被看成是一个数字系统。没有控制部件又不能按照一定程序进行操作的系统只能看成是一个逻辑功能部件或子系统。

11.1.2 数字系统的基本结构

数字系统可由多个功能模块或子系统组成。按照其作用性质,数字系统在结构上可分为两部分:一部分是用来实现信息传送和加工处理的数据处理单元即处理器;另一部分是产生控制信号序列的控制单元即控制器,如图 11.1.1 所示。控制单元是根据外部控制信号以及反映数据处理单元当前状况的状态信号,发出对数据处理单元的控制序列信号;在此控制序列信号作用下,数据处理单元对输入信息(数据)进行分解、组合、传输、存储和变换,产生相应的输出信息(数据),并且向控制单元输出状态变量信号,用以表明数据处理单元当前的工作状态和处理数据的结果。控制单元在收到状态变量后,再决定发出下一步的控制序列信号,使数据处理单元执行新一轮的一组操作。

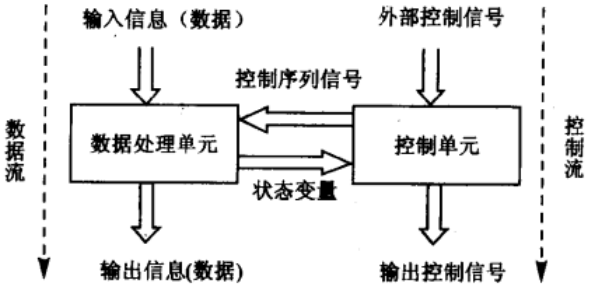


图 11.1.1 基本数字系统结构

数据处理单元和控制单元是一个数字系统中最基本的两部分。尽管各种数字系统可能具有完全不同的功能和形式,但是都可以用数据处理单元和控制单元所构成的数字系统的基本结构来描述。控制单元产生的输出控制信号影响着其他系统控制单元的操作,使本系统与其他系统协调一致地工作。控制单元的外部控制信号也可能是其他系统的输出控制信号。数字系统中就是这样通过数据处理单元和控制单元之间的密切配合、协调工作,成为一个自动实现信息处理功能的有机整体。

11.1.3 数字系统设计的特点

随着科学技术的发展,数字系统(如计算机系统)已达到前所未有的复杂程度和技术水准。采用 LSI、VLSI 工艺制造的微处理器、单片机、ROM、RAM 和 PLD 子系统模块,已经成为数字系统设计中的基本构件。基于经典开关理论,追求门电路和输入项最小化的传统设计,已经不能适应新的情况,因此在现代数字系统设计中层次结构化和模块技术显得非常重要。

1. 系统的层次结构化

系统学的一个重要的观点是:系统是分层次的,是研究复杂对象的总称。系统是若干相互依赖、相互作用完成特定功能的有机整体。数字系统可以认为是一种层次结构,其设计过程是:以用户对系统性能的要求所定义的系统功能说明为出发点,根据系统结构的观点确定系

统内包含的数据流和控制流，自上而下将系统逐级分解为可由 LSI，VLSI 等硬件和软件实现的模块。然后通过逻辑设计选择合适的结构和物理实现途径，将元器件和基本构件集成为实现某种功能或性能的模块和子系统，由模块和子系统组装成系统，实现自下而上的组装和调试。

数字系统设计过程分为图 11.1.2 所示的四个层次：性能级、功能级、结构级和物理级。将性能级的说明映射为功能级的设计过程称为系统设计；将功能级的描述转换为结构（逻辑）的过程称为逻辑设计；将逻辑结构转化为物理级（电路）上的实现称为物理设计。

（1）性能级

要求设计者集中精力研究分析用户让系统“做什么”，明确设计什么，达到什么指标。以系统说明书的形式作为设计者与用户之间的合同，避免设计过程中不必要的反复，保证设计顺利进行，从而为进一步的系统设计、逻辑设计、物理设计以及最后测试、验收提供依据。

对系统性能要求即用户要求，可以用多种描述形式来正确说明，如文字、图形、符号、表达式以及类似于程序设计的形式语言等。为了精确地、无二义性地描述用户要求，系统说明书力求简明易懂，尽量避免专业技术的概念、术语、具体的实现方法和技术细节；反复检查，尽早发现并纠正潜在的错误和缺陷。

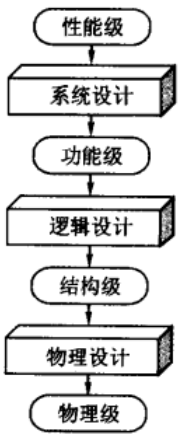


图 11.1.2　系统层次结构

（2）功能级

功能级又称为系统结构级。设计者从系统的功能出发，把系统划分为若干子系统（或模块）；每个子系统又可以分解为若干模块（或子模块）；子系统（或模块）间通过数据流和控制流建立起相互之间的联系。从而给出系统的总体结构即系统设计。

随着系统结构分解过程的推移，每个子系统（或模块、子模块）的功能越来越专一，越来越明确，总体结构越来越清晰。在结构设计中应该采用合适的手段和方法（如硬件描述语言、结构图）对子系统（或模块）以及子系统（或模块）之间的逻辑关系加以描述和定义。例如，利用可编程逻辑器件设计数字系统的顶层文件即为系统的整体设计。

（3）结构级

结构级又为逻辑级，是将子系统（或模块）的功能描述转化为实现子系统（或模块）功能的具体硬件和软件的描述。对子系统（或模块）的功能首先进行算法设计，把其功能进一步分解、细化为一序列运算和操作，然后采用多种描述方式如算法流程图、ASM 图、寄存器传送语言、HDL 语言、逻辑表达式和逻辑图等来描述其运算和操作，进行逻辑设计。

（4）物理级

物理级也为电路级。它把上一步描述功能的算法转换成逻辑电路或基本逻辑构件的物理实现，包括元器件、集成芯片的选择；电路布线、布局和优化；电路测试、电源及抗干扰措施的实现等等。随着 VLSI 和电子设计自动化（EDA）的发展，越来越多的系统采用 LSI 和 VLSI 芯片（如 ROM，PLD，CPU）作为电路设计的基本构件，并且利用 EDA 技术，使系统设计大大简化，系统实现变得容易，系统具有较大的冗余度，降低设计周期和成本。改变了物理设计的设计思想和设计方法。

任何复杂的数字系统可以最终分解成基本门和存储元件，由大规模集成电路来实现。集成电路设计过程就是把高级的系统描述最终转换成如何生产芯片的描述过程。设计过程中的层次化、结构化使得设计能力有了很大提高。层次化的设计方法能使复杂的电子系统简化，并能在不同的设计层次及时发现错误并加以纠正；结构化的设计方法中把复杂抽象的系统划分成一些可操作的模块，允许多个设计者同时设计一个系统中的不同模块，而且某些子模块的资源可以共用。

电路的层次化、结构化设计过程可以用 Gajski 于 1983 年提出的 Y 图来描述，如图 11.1.3 所示。图中三个轴表示三个互不相同的设计域：行为域、结构域和物理域。行为域描述一个特定的系统做些什么，要完成的功能；结构域描述实现某一功能的具体结构以及各个组成部件是怎样连接在一起的；物理域描述结构的物理实现，即怎样实际制造一个满足一定连接关系的结构和功能的芯片。

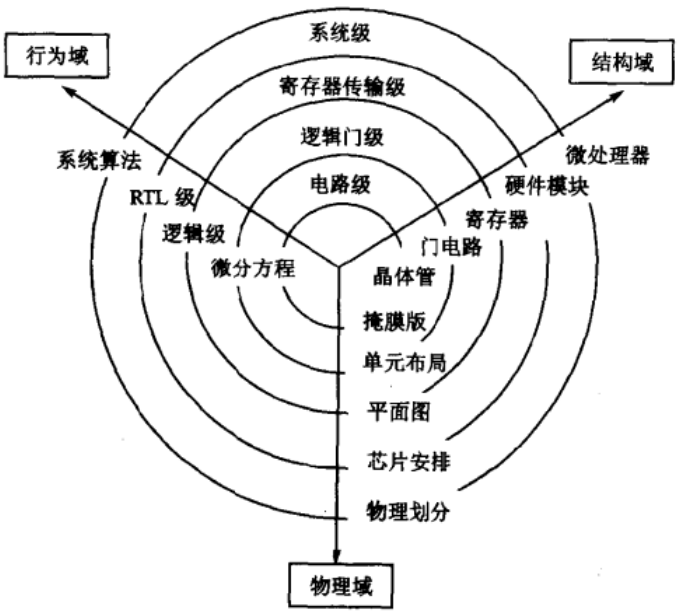


图 11.1.3 电路层次化设计的 Y 图描述

每一个域都可以在不同的抽象层次上描述，离中心越远则抽象程度越高。电路设计的总过程是沿每个轴向中心逼近的一个序列。从行为到结构、到物理实现的迭代，然后回到更低一级部件的行为。随着设计的进行，迭代的螺旋指向 Y 图的中心，得到最后的掩膜版设计。这些抽象层次中最高层次为系统级，最低层次为版图级或称物理级，如表 11.1.1 所列。

当前 ASIC 的设计就可以说是硬件的一种描述形式到另一种描述形式的转换过程。由寄存器传输级(RTL)的行为描述转换成下一级(逻辑门级)的结构描述(用逻辑门、触发器)，称为逻辑综合；由结构域的描述转换成物理域的描述称为物理综合。这些综合技术是电子设计自动化中的关键技术。除了从系统级的自然语言行为描述到系统级或芯片级的结构描述的转换之外，其他各种描述形式的变换都有了自动综合工具或 EDA 工具。这些工具为电路设计者提供了很大的帮助，也改变了电子系统的设计方法。

表 11.1.1　电路设计的层次描述

设计层次	行为描述	结构描述	设计考虑
系统级	自然语言描述的性能、指标	方框图	系统功能
芯片级(IC 中也称系统级)	算法	微处理器、存储器、通用集成电路等组成的方框图	时序、同步、测试
寄存器级(IC 中为宏单元)	数据流图、有限状态机、状态表、状态图	寄存器、ALU、计数器、MUX、ROM 等	时序、同步、测试
逻辑门级	布尔方程、卡诺图、Z 变换	逻辑门、触发器	选择适当的基本门
电路级	电压、电流的微分方程	晶体管、R、L、C 等	电路性能、延时、噪声
版图级	几何图形与工艺规则		

2. 系统设计中的模块化技术

模块化技术就是将系统总的功能分解成若干个子功能，通过仔细定义和描述的子系统来实现相应的子功能。子系统又可以分解为若干模块或子模块，随着分解的进行，使抽象的功能定义和描述为具体的实现提供更多的细节，从而保证系统总体结构的正确。

一个系统的实现可以有许多方案，划分功能模块也有多种模块结构。结构决定系统的品质，这是系统论中的一个重要观点，即一个结构合理的系统可望通过参数的调整获得最佳的性能；一个不合理的系统结构即使精心调整，也往往达不到预定的效果。因此系统整体结构方案的设计直接关系到所设计系统的质量。在划分系统的模块结构时，应考虑以下几方面：

① 如何将系统划分为一组相对独立又相互联系的模块；

② 模块之间有哪些数据流和控制流信息；

③ 如何有规则地控制各模块交互作用。

模块结构的相对独立性从两个方面来衡量：一方面是指模块内各元器件组成的部件或构件之间联系的紧密程度；另一方面是指模块之间的联系程度。提高模块内部的紧密程度和降低模块之间的联系，是提高模块相对独立性的两个方面。如果把系统中密切相关的组件或构件划分在不同的模块中，则使其内部的凝聚度降低，模块之间联系程度提高。这对系统的理解、设计、实现、调试和修改都带来许多困难。因此为了设计一个易于理解和开发的系统结构，应该提高模块相对独立性。

描述系统模块结构的方法主要有以下两种：

① 模块结构框图。以框图的形式表示系统由哪些模块(或子系统)组成以及模块(或子系统)之间的相互关系，定义模块的输入/输出信息和作用。

② 模块功能说明。采用自然语言或专用语言，以算法形式描述模块的输入/输出信号和模块的功能、作用及限制。

由于系统中的模块具有相对独立性、功能比较专一，对其中的数据处理单元和控制单元可以单独描述和定义，通过逻辑设计最终达到物理实现。每个模块还可单独进行测试、排错和修改，使复杂的设计工作简单化，提高研制工作的平行性。同时限制了局部错误的蔓延和扩散，提高了系统的可靠性。

11.2 数字系统设计方法

11.2.1 试凑设计法

试凑设计就是用试探的方法把系统的功能要求分成若干个相对独立的功能模块，选择合适的功能部件拼接组合起来，构成一个完整的数字系统。试凑法主要是凭借设计者对逻辑设计的熟练、技巧和经验来确定系统结构方案，划分模块，选择器件，以电路结构图的形式拼接模块。对于一些规模不大，功能不太复杂的数字系统，选用中、大规模集成器件，采用试凑设计法，具有设计过程简单、逻辑关系清晰、电路调试方便、性能稳定可靠等特点，目前还被广泛采用。

试凑法并不是盲目的，通常按以下步骤进行。

(1) 分析系统要求，拟定系统总体方案

分析设计任务书，明确系统功能。确定系统有哪些输入、输出信息，它们的特征、格式和传送方式，以及系统需要完成的处理任务等。

(2) 划分功能模块，建立总体结构框图

划分功能模块可采用由粗到细的方法，先将系统分为处理单元和控制单元，再按处理任务或控制功能逐一划分。功能模块的大小要适当，以功能比较单一、易于实现和方案比较来划分模块。

(3) 选择并构成各功能部件

将上面功能模块进一步分解成为若干相对独立的子模快(功能部件)，以便直接选用中、大规模集成器件来设计和实现。

(4) 电路的实现

连接各个模块，绘制整体的逻辑电路图，综合考虑各功能模块之间的时序、信号传送和控制等逻辑关系。最后绘制印刷电路板并安装调试。

11.2.2 自顶向下的设计方法

近几年来，数字系统设计方法发生了较大变化，由自底向上(Bottom - up)的设计方法变为自顶向下(Top - down)的设计方法。过去设计的基本思路一直是先选用标准通用集成电路芯片，再由这些芯片和其他元件自下而上地构成电路、子系统和系统。这样设计的系统所用元件的种类和数量较多，体积、功耗大，修改困难，可靠性差。

自顶向下的设计方法采用系统层次结构，将系统的设计分成几个层次进行描述。通常把系统总的技术指标的描述称为性能级或系统级的描述，这是最高一级描述。由此导出实现系统功能的算法即系统设计。根据算法把系统分成若干功能模块(子系统)，每个模块又分解为几个子模块、用逻辑框图形式描述各功能模块(子系统)的组成和相互联系，设计出系统结构框图，这一级称为功能级描述。最后进行逻辑设计，详细给出实现系统的硬件和软件描述，被称为电路级描述。

自顶向下的设计方法是一种由抽象的定义到具体的实现、由高层次到低层次的转换逐步求精的设计方法。其设计过程并非是一个线性过程，在下一级的定义和描述中往往会发现上

一级定义和描述中的缺陷或错误，因此必须对上一级中的缺陷或错误进行修正，使其更真实地反映系统的要求和客观的可能性。整个设计过程是一个“设计—验证—修改设计—再验证”的过程。

数字系统的制作和测试通常是按系统设计的相反顺序进行，即自底向上的集成过程。它是从具体的器件和部件开始，逐步由下而上组装和集成为完成某局部功能的模块（或子系统），最后由这些模块构成一个完整的数字系统。但是组成的系统总体结构有时不是最佳的。可以这样说，数字系统的自顶向下的设计方法反映了人们从预定的目标出发，不断探索、认识和不断深化的过程，而自底向上的集成过程则是通过局部的、较简单的功能模块（或子系统）的经验积累，以达到系统预定目标要求的实践过程。

11.3　算法流程图及 ASM 图

自顶向下的设计过程实际上是不同层次的描述形式间的变换，因此对系统进行描述的问题将贯穿设计的全过程，在不同的设计阶段采用适当的描述方式。在系统结构设计阶段常用的描述方式有方框图、定时图和算法流程图。正确地定义和描述设计目标的功能和性能，是设计工作正确实施的依据，是进一步设计的基础。

11.3.1　方框图和定时图

方框图是系统设计阶段最常用、最重要的描述手段。它可以详细描述数字系统的总体结构，并作为进一步设计的基础。方框图不涉及过多的技术细节，具有直观易懂、系统结构层次化和清晰度高、易于方案比较以达到系统总体优化等优点。

方框图中每一个方框（矩形框）定义一个信息处理、存储或传送的子系统或模块，在方框内用文字、表达式、通用符号和图形来表示该子系统或模块的名称或主要功能。方框之间采用带箭头的直线连接，表示各子系统或模块之间数据流或控制流的信息通道，箭头指示信息的传输方向。

一般总体结构方框图需要有一份完整的系统说明书，在说明书中不仅需要给出表示各子系统或模块的方框图，同时还需给出每个子系统或模块功能的详细描述。

数字系统中无论是信号的采集、传输、处理或存储，都是在特定的时间意义上的操作，是严格按照时序进行协调和同步的。系统中每个子系统或模块的功能正是体现按规定的时标实现输入信号向输出信号的正确转换。定时图（时序图）用来定时地描述系统各模块之间、模块内部各功能部件之间以及部件内各门电路或触发器之间输入信号、输出信号和控制信号的对应时序关系及特征（时钟信号为电平或脉冲、同步或异步）。

定时图的描述是逐步深入细化的过程，由描述系统输入/输出信号之间关系的定时图开始。随着系统设计的深入，定时图不断地反映新出现的系统内部信号的定时关系，直到对系统内各信号时序关系的完全描述。在系统进行功能和时序测试时，可借助 EDA 工具，建立系统的仿真波形文件，通过仿真来判定系统中可能存在的问题；在硬件调试和运行时，可通过逻辑分析仪或示波器对系统中节点处的信号进行观察测试，以判定系统中可能存在的错误。

11.3.2 算法流程图

算法流程图是用特定的几何图形(矩形、菱形、圆形)、指向线和简单文字说明,来描述数字系统的基本工作过程,是描述数字系统功能的最常用方法之一。它与软件设计中的流程图十分相似。

1. 基本符号

算法流程图常使用工作块、判别块、条件块、入口出口块,如图 11.3.1 所示。

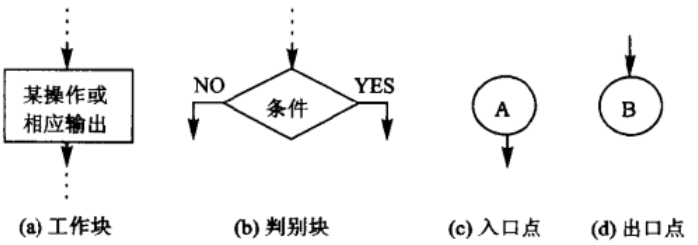


图 11.3.1 算法流程图基本符号

① 工作块:是一个矩形块,块内用简要的文字来说明应进行的一个或者若干个操作以及相应的输出。

② 判别块:其符号为菱形,块内给出判别变量及判别条件。判别条件是否满足,决定系统将进行不同的后续操作。图 11.3.2(a) 中判别变量是 CNT,判别条件为 CNT=24 时,计数器清零(置零),否则计数器进行加 1 计数。有时有多个判别变量,从而可能构成两个以上的分支,如图 11.3.2(b) 所示。

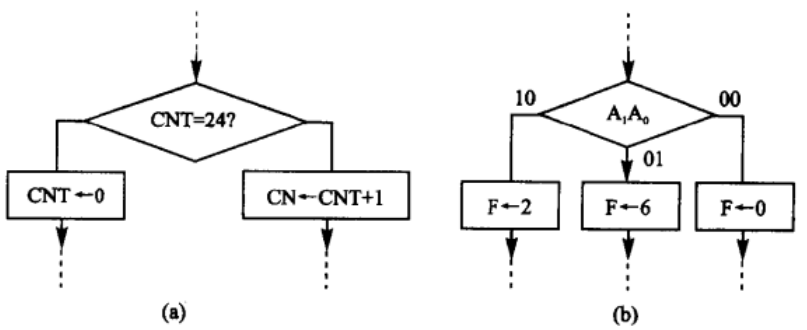


图 11.3.2 判别块举例

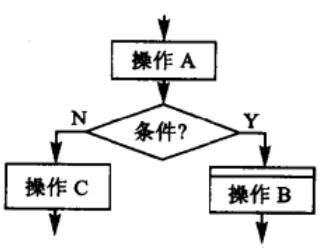


图 11.3.3 条件块举例

③ 条件块:为一个带横杠的矩形块,总源于判别块的一个分支。条件块中的操作与特定的条件有关,因此被称为条件操作。工作块规定的操作无前提条件,是独立的操作。条件块是算法流程图所特有的,也是与软件流程图的主要区别之一。图 11.3.3 中操作 A 和操作 C 均为工作块的操作;操作 B 为条件操作,仅是操作 A 的延伸。从时序上看,操作 B 有可能与操作 A 同时进行,在满足条件时,操作 B 与操作 A 是在同一时钟周期内进行操作;而操作 C 则

只可能在操作 A 完成后才进行。

④ 入口出口块：用圆形符号表示。入口点指明算法的起点或算法的继续点，当算法太长，一页写不完另起一页时，就需要一个继续点。有入口点就应有出口点。

2. 算法流程图的建立

算法流程图可以描述整个数字系统对信息的处理过程以及控制单元所提供的控制步骤。流程图的建立也就是算法设计过程。它是把系统要实现的复杂运算或操作分解成一系列子运算或操作，并且确定执行这些运算或操作的顺序和规律，为逻辑设计提供依据。

由于系统的逻辑功能多种多样，至今尚无从系统功能导出算法的通用方法和步骤。设计者需要仔细分析设计功能要求，将系统分解成若干功能模块，把要实现的逻辑功能看作是应进行的某种运算或操作。用算法流程图来描述时通常具有两大特征：

① 包含若干子运算或操作，实现数据或信息的存储、传输和处理；

② 具有相应的控制序列，控制各子运算或操作的执行顺序和方向。

下面通过例子说明算法流程图的建立过程。

例 11.3.1　试设计一个对串行输入信号含 1 的统计电路，统计输入信号中包含 1 的个数，其中输出信号 $Z=z_{m-1}z_{m-2}\cdots z_0$ 表示统计个数，串行输入信号为 $X=x_{n-1}x_{n-2}\cdots x_0$。

对于这样一个简单的逻辑问题，却难以用数字逻辑电路中的状态表对它进行描述。对于长度为 n 位的输入信号，将有 2^n 种不同的组合，当 $n>7$ 时显然状态表将变得十分庞大。因此用状态表或状态图来描述的方法并非适合所有逻辑问题。但是可以把这个逻辑问题分解为若干操作，用算法流程图来描述其算法过程。

首先为了统计输入信号中含有 1 的个数，该统计电路必须有一个加 1 计数的操作，以及累计输入信号位数的操作。另外考虑到仅当输入信号为 1 时才进行统计 1 的计数操作，故还应有判别操作(控制电路)。

通过对设计要求的分析，画出其框图如图 11.3.4 所示。其中 ST 为开始标志输入信号，END 为统计结束标志信号，假设输入信号长度为 15，那么对应的算法流程图如图 11.3.5 所示。当 ST 有效时(ST=1)，首先将 Z(计数器)和 n(计数器)清零。然后逐个判断输入 X 是否为 1，若为 1，则 Z 加 1。当 n=15 时，表示一组输入序列信号结束。END=1 表示一组输入序列的统计个数 Z 有效，然后回到等待状态，为下一组输入序列信号作准备。

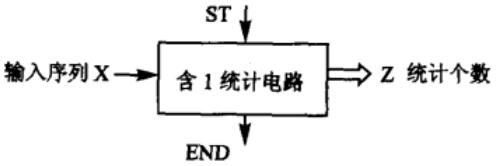


图 11.3.4　含 1 的统计电路框图

例 11.3.2　某单位有一台备用的柴油交流发电机，该机在市电停电时立即自动发电，在启动后三分钟内测量发电机的转速，如果转速没有达到规定值则报警。在进入正常发电状态时，需要不断测量转速和输出电压，以此调整供油量，保证发电机按规定要求输出交流电。如果转速或输出电压发生异常则报警，并且在三分钟内停机。试设计该发电机的控制电路。

详细分析设计要求和功能，分解为若干操作，并且按照控制过程的顺序和规则来实施这些操作。由此可直接画出如图 11.3.6 所示的发电机的控制电路的算法流程图。

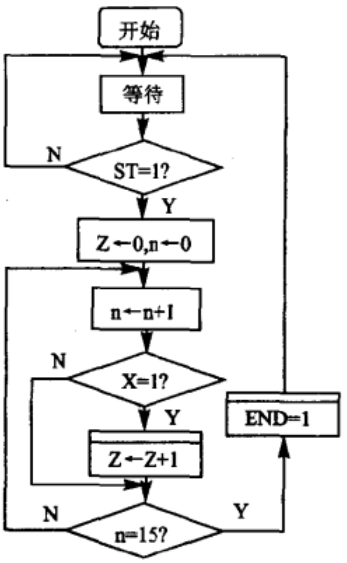


图 11.3.5 含 1 的统计电路算法流程图

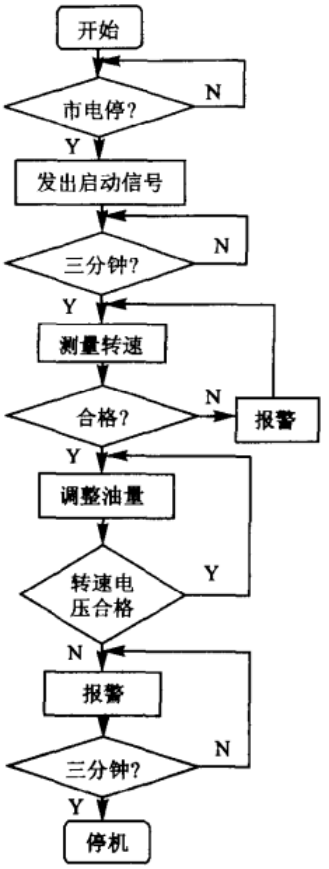


图 11.3.6 发电机控制电路算法流程图

11.3.3 ASM 图

算法流程图只是按照操作所规定的先后顺序排列的步骤描述，并未严格地规定完成各操作所需的时间及操作之间的时间关系。因此不能直接由算法流程图得到下一步逻辑设计，必须把算法流程图转换成 ASM 图(算法状态机图)、MDS 图(备有记忆文件的状态图)或状态表，作为下一步逻辑设计的依据。

ASM 图采用类似于流程图的形式来描述控制器在不同的时间内应完成的一系列操作，反映了控制条件及控制器状态的转换。这种描述方法与控制器硬件实施有很好的对应关系。

1. ASM 图的基本符号

ASM 图是硬件算法的符号表示法，可方便地表示数字系统的时序操作。它由四个基本符号组成，即状态框、判断框、条件框和指向线。

① 状态框：用一个矩形框来表示控制器的一个状态。该状态的名称和二进制代码(已状态分配)分别标在状态框的左、右上角；矩形框内标出在此状态下数据处理单元应进行的操作以及控制器的相应输出，如图 11.3.7 所示。其中图(b)说明控制器处于 S3 状态(编码为 011)时，执行寄存器清零的操作，发出输出信号 C，且高电平有效。有时框内的操作可略去，仅说明输出信号。

② 判断框：用菱形表示状态在条件转移时的分支途径，判断变量（分支变量）写入菱形框内作为转移条件，在判断框的每个转移分支处写明满足的条件，如图 11.3.8(a) 所示。判断框中的判断变量并不局限于一个，可以有多个判断变量和多条分支途径。

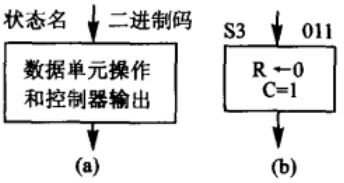


图 11.3.7　状态框

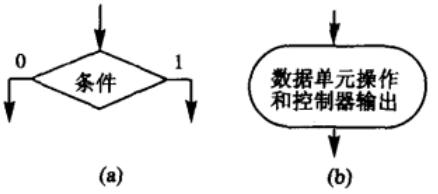


图 11.3.8　判断框与条件框

③ 条件框：用椭圆框表示，框内标出数据处理单元的操作以及控制器的相应输出，如图 11.3.8(b) 所示。条件框一定是与判断框的一个转移分支相连接，仅当判断框中判断变量满足相应的转移条件时，才进行条件框中表明的操作和信号输出。虽然条件框和状态框都能执行操作和输出信号，但两者之间有很大区别。图 11.3.9 给出一个条件框的实例，当系统处于 S1 状态下，并且变量 A 满足条件 A=1 时，立刻执行寄存器 R 清零操作（在 S1 状态下），然后进入 S3 状态。如果变量 A 不满足条件 ($\bar{A}$=1)时，将进入 S2 状态执行计数器 F 的加 1 操作，然后在下一个时钟到达时才进入 S3 状态。

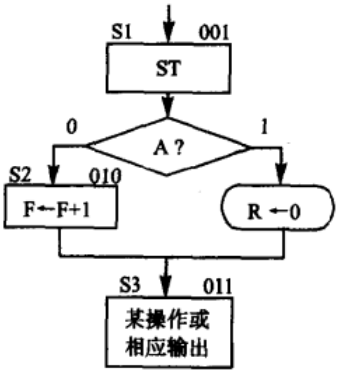


图 11.3.9　条件框例子

④ 指向线：用箭头线表示，用于把状态框、判断框和条件框有机地连接起来，构成完整的 ASM 图。

2. ASM 块

ASM 图可以细分为若干个 ASM 块，每个 ASM 块必定包含一个状态框（必有），可能还有几个同它相连接的判断框和条件框。一个 ASM 块只有一个入口和由判断框构成的几个出口。仅包含一个状态框，无判断框和条件框的 ASM 块是一个简单块，如图 11.3.10 所示。每个 ASM 块表示一个时钟周期内系统所处的状态，在该状态下完成块内的若干操作。ASM 块中的状态框和条件框的操作，是在一个共同的时钟周期内（即某个状态下）一起完成的。并且在下一个时钟周期内使现状态转移到新状态，进入另一个 ASM 块。

ASM 图类似于状态图，一个 ASM 块等效于状态图中的一个状态。判断框表示的判别条件相当于状态图定向线旁标记的判断变量的取值（二进制码）。如把 ASM 图转换成状态图，就可以利用时序逻辑电路的设计步骤来设计系统控制器。图 11.3.11 是 ASM 块（图 11.3.10）转换的状态图。状态图虽然可以表示状态的转移、转移条件和输出信号，但是它无法表示操作和条件输出。这正是状态图与 ASM 图的差别。状态图只能定义一个控制器，而 ASM 图除了定义一个控制器以外，还指明了被控制的数据处理单元中应实现的操作，所以 ASM 图定义的是整个数字系统。

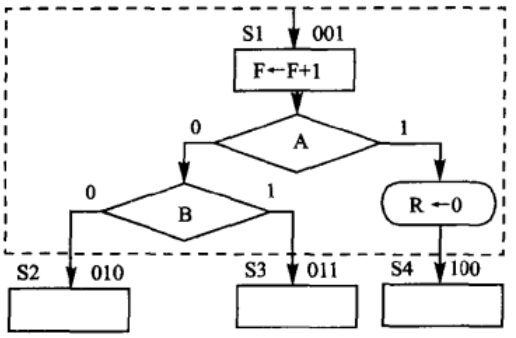


图 11.3.10 ASM 块

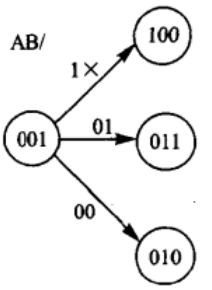


图 11.3.11 状态图

3. 由算法流程图导出 ASM 图

ASM 图和算法流程图之间有一定的对应关系，两者之间的工作块和状态框、判别块和判断框、条件块和条件框都基本对应。确切地说，算法流程图规定了系统应进行的操作及操作的顺序；ASM 图规定了为完成这些操作及操作的顺序所需的时间和控制器发出的输出信号。由算法流程图导出 ASM 图，主要是定义状态，其原则有三条，即

① 在算法起点定义一个初始状态。

② 必须用状态来分开不能同时实现的操作。例如 F←F+1 和 F←0 两个操作，寄存器 F 不能同时完成加 1 和清零两个操作，因此两操作必须分两步进行，即用状态来分开。

③ 判断框中的条件如果受寄存器操作的影响，则应在它们之间安排一个状态。如图 11.3.12(a) 所示为算法流程图。该图的操作顺序是根据 A 加 1 后，判断 A 是否等于 n，算法执行两个分支中的一个。用 ASM 图来表示该算法时，应在操作和判断框之间定义一个状态，如图11.3.12(b)所示，否则检测的是 A 加 1 之前的值。

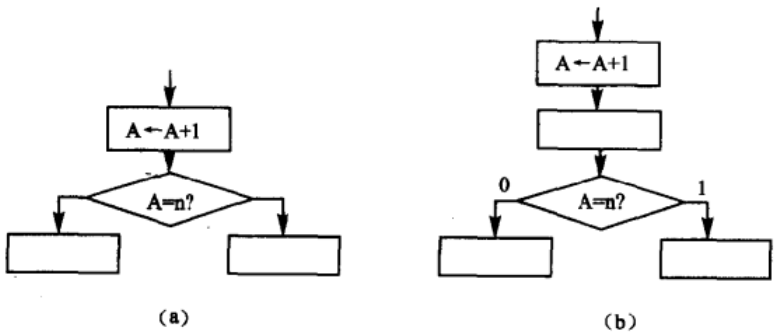


图 11.3.12 算法流程

例 11.3.3 一个数字系统的数据处理单元有两个触发器 E 和 F 及二进制计数器 A，计数器的各位为 A4，A3，A2 和 A1，启动信号 ST 使计数器 A 和触发器 F 清零，从下一个时钟脉冲开始加 1 计数，直到系统停止工作为止。A4 和 A3 的值决定系统的操作序列，即

A3＝0，触发器 E 清零，并继续计数；

A3＝1，触发器 E 置“1”，并检验 A4。A4＝0，计数；A4＝1，触发器 F 置“1”，停止计数。

此例题的算法流程图如图 11.3.13 所示，ASM 图如图 11.3.14 所示。

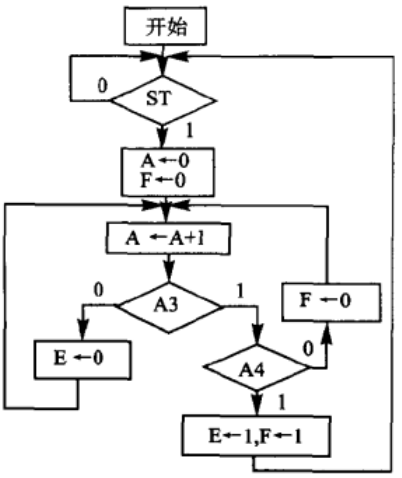


图 11.3.13　算法流程图

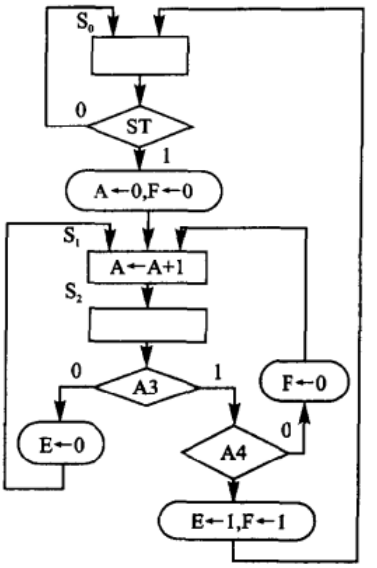


图 11.3.14　ASM 图

例 11.3.4　对图 11.3.4 含 1 统计电路框图先分成两部分：控制器和数据单元(计数器 A,B)，如图 11.3.15 所示；再根据图 11.3.5 的算法流程图，绘出含 1 统计电路控制器的 ASM 图，如图 11.3.16 所示。

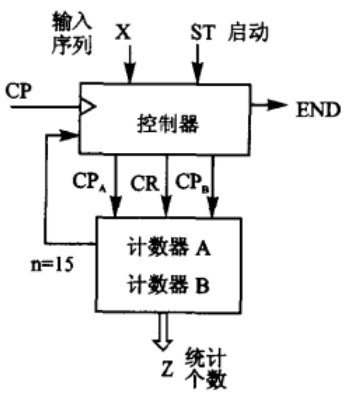


图 11.3.15　含 1 统计电路的结构图

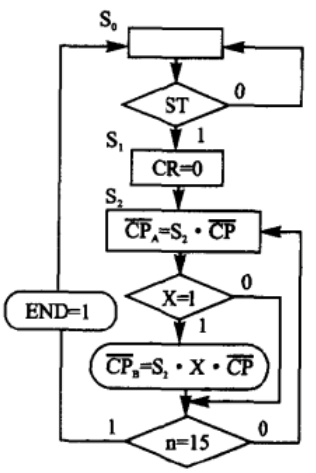


图 11.3.16　含 1 统计电路的 ASM 图

第十二章　数字系统的实现

数字系统设计在经过系统设计阶段，确定了系统的算法结构，并导出用 ASM 图表示的相应算法以后，面临的任务是逻辑设计，即通过硬件和软件设计来实现系统的功能。目前，实现系统的途径主要有以下几方面：

① 以标准通用的 SSI，MSI 和 LSI 集成器件来构成；

② 以单片微处理器为核心；

③ 将整个系统配置在一片或数片 PLD 芯片内，辅以必要的辅助器件，在固化于存储器内的软件控制下实现系统功能；

④ 研制相应的 ASIC，构成单片系统。

在这四种方法中，第一种方法是最经典的方法，现被国内广大设计者所采用。第二种方法的价格便宜、易实现，适用于运行速度要求不高的场合，也得到广泛应用。随着集成电路制造技术的发展，近年来出现一系列性能更为优越的高密度 PLD，使第三种方法越来越显示出其潜力和优越性：价廉、运行速度高、体积小、易于重复修改设计等。第四种方法是设计者面临的新技术和新挑战，将得到越来越多的应用。本书主要讨论用 MSI，LSI 和高密度 PLD 来实现数字系统的方法。

根据前面章节所述，数字系统可由两大部分组成，即数据处理单元和控制单元。逻辑设计过程就是完成控制单元和数据处理单元的设计和实现。

12.1　数据处理单元

数据处理单元又称受控电路。它由寄存器和组合电路组成，寄存器用于暂存信息；组合电路实现对数据的加工和处理。数据处理单元的结构示于图 12.1.1，输入信号（数据）$X(X_1, X_2, \cdots, X_i)$和输出信号（数据）$Z(Z_1, Z_2, \cdots, Z_n)$表示通过数据处理单元的数据。控制信号 $T(T_1, T_2, \cdots, T_r)$是控制器发出的命令信号，决定在时钟脉冲出现时，数据处理单元应完成什么操作。输出信号 $E(E_1, E_2, \cdots, E_k)$是由控制信号 T 形成的，加在寄存器的功能控制端，实现寄存器操作的功能选择。状态变量信号 $C(C_1, C_2, \cdots, C_m)$是数据处理单元产生的信号。它反馈给控制器，决定下一个操作步骤。

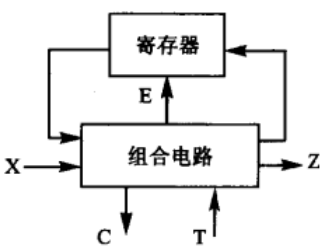


图 12.1.1　数据处理单元的结构

在算法流程图或 ASM 图中已给数据处理单元规定了明确的逻辑功能，这些功能概括为数据存储、算术和逻辑运算、数据传送和变换等。要实现数据处理的功能，可以通过集成电路制造厂商提供的多种规格的通用集成电路芯片，或者由硬件描述语言来设计。

12.1.1　数据处理单元设计的基本步骤

数据处理单元的设计过程是根据算法流程图或 ASM 图，求出数据处理单元要完成的一系列运算和操作，列出明细表即规定数据处理任务的表格。明细表由操作明细表和状态变量表两部分组成。根据数据处理单元的明细表，选择能完成这些操作的集成电路芯片，进行电路连接。

采用通用集成电路芯片进行数据处理单元设计时，其基本步骤如下。

1. 确定数据处理单元的逻辑框图

算法流程图的形成过程就是数据处理单元结构的建立过程，根据算法流程图和结构选择方案，画出数据处理单元的逻辑框图，并由此明确它与控制单元之间必须交换的信息以及这些信息之间的时间关系。

2. 列出数据处理单元的明细表

提出对数据处理单元的全面要求，建立数据处理单元的技术规范——明细表，确定在每一控制信号作用下完成的一组操作；确定处理信息时要完成哪些信息检验；确定输出信息是什么。根据算法可以建立数据处理单元的明细表，即操作明细表和状态表。操作明细表定义了数据处理单元，状态表定义了控制器。

3. 器件选择

采用通用集成电路芯片设计数据处理单元时，应该注意两条：一是易于控制，即器件的控制方式、控制信号以及产生这些控制信号的逻辑应尽可能简单，以简化控制单元的设计；二是力求模块数少，以减少电路体积、功耗，降低成本。

选择具体的集成电路器件由设计者的经验和技巧所决定。除了常用的各种 SSI，MSI 或 LSI 数字集成电路外，有时还需配置多种辅助电路，如脉冲电路；还会遇到 A/D 和 D/A 转换器、集成运算放大器、锁相环以及其他辅助器件。

12.1.2　数据处理单元设计的实例

例 12.1.1　按照例 11.3.3 的题意和图 11.3.14 的 ASM 图，设计系统的结构框图如图 12.1.2 所示。数据处理单元的明细表如表 12.1.1 所列。

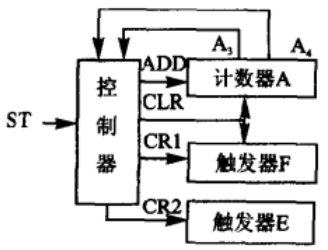


图 12.1.2　设计系统结构框图

表 12.1.1　数据处理单元的明细表

操作表		状态变量表	
控制信号	操　作	状态变量	定　义
NOP	无操作(等待)		
CLR	F←0，A←0(清零)	C_1	ST
ADD	A←A+1	C_2	A_3
CR1	F←1(置"1")	C_3	A_4
CR2	E←1(置"1")		

明细表包含两个子表：一个是操作表；另一个是状态变量表。操作表列出在控制信号作用下，数据处理单元应实现的操作和产生的输出；状态变量表定义数据处理单元输出的状态变

量。在表 12.1.1 中 NOP 表示控制器处于等待状态，处理单元无操作，等待启动信号 ST 的到来。控制器发出的一个控制信号实现数据处理单元相应的一组操作，然后控制器根据处理单元输出的状态变量决定下一步发出的控制信号，直到工作完成为止。

例 12.1.2 试设计含 1 统计电路的数据处理单元。

由图 11.3.15 所示的逻辑结构图可知，该电路的数据处理单元由两个计数器(计数器 A，B)构成。因输入信号序列长度为 15，所以选用两个四位二进制计数器 74161。其数据处理单元原理图如图 12.1.3 所示。计数器 A 负责对输入信号序列长度的计数，反馈信号 n=15(取自计数器 CO 端)。计数器 B 的输出 Q_3，Q_2，Q_1，Q_0 即为统计结果。

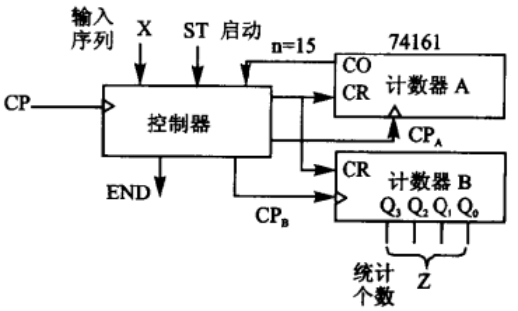


图 12.1.3 含 1 统计电路的数据处理单元

例 12.1.3 设计八位串行数字密码锁的数据处理单元。锁串行接收输入数码，当数码的位数和位值与开锁密码相同时，锁被打开，否则锁不开。

假设数字锁每次接收输入数码为 0 或 1，每当按下“读码”按钮 READ 时，输入数码送入系统。开锁密码事先存入锁内，只有输入数据与开锁密码完全相同时，锁才进入开锁状态，接着在开锁信号 TRY 作用下，锁才被打开。其系统结构图如图 12.1.4 所示。数据处理单元由四部分构成，即

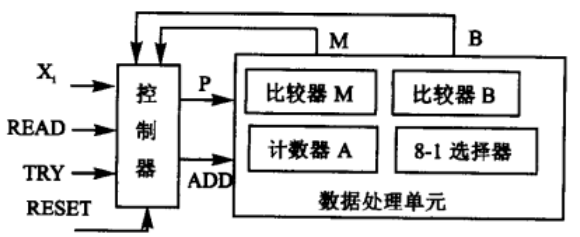


图 12.1.4 八位串行数字锁结构图

计数器 A：记录输入数据的次数；

位数比较器 M：比较输入数据的次数和设置的参考数，若相同则 M=1，否则 M=0；

位值比较器 B：比较输入数据的值(P)，若与对应位开锁密码的值(D_i)相同，则比较器 B=1，否则 B=0；

数据选择器 8-1 MUX：八选一数据选择器输入端(D_0～D_7)设置为开锁密码。

根据题意导出算法，建立 ASM 图，如图 12.1.5 所示。在复位信号 RESET 作用下，进入初始状态 T_0，计数器 A 清零，然后进入接收数据状态 T_1。在 T_1 状态下，首先检测开锁信号 TRY 是否有效。如果开锁者不知道数字锁的位数，会错误地发出 TRY 信号，使系统进入错

误状态 T_3。若 TRY 无效，而读码信号 READ 有效，则根据位值相等信号 B 判断输入数码的位值是否正确。若 B 无效，则系统进入错误状态 T_3；若 B 有效，则检测位数相等信号 M。若 M 有效，则系统进入开锁状态 T_2；若 M 无效，则说明位数不够，应继续接收输入数码。每正确接收一次数码，计数器 A 加 1，正确接收 8 次数码后，M 有效(M=1)。在开锁状态 T_2 下，如果开锁者继续输入数码，则表示输入数码已超过设定次数(如 8 次)，系统进入错误状态 T_3。在 T_2 状态和开锁信号 TRY 作用下，锁被打开(T_4)。

数字密码锁处理单元的电路原理图如图 12.1.6 所示。

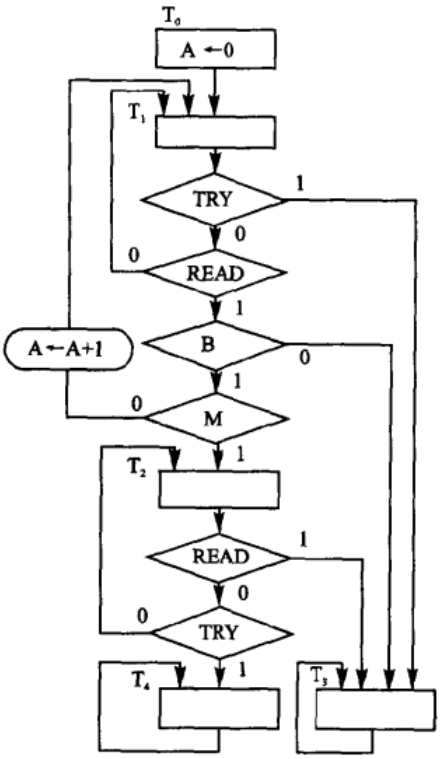


图 12.1.5　八位串行数字锁的 ASM 图

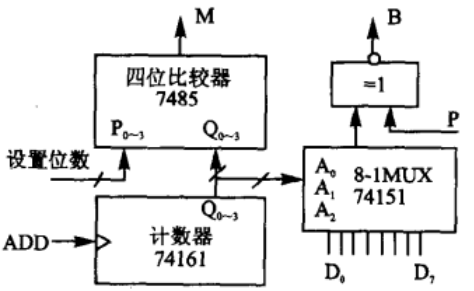


图 12.1.6　数字密码锁电路原理图

12.2　控制单元的设计

数据处理单元正确有序地运算和操作是在控制单元的正确有序地管理下进行的。控制器在每一个计算步骤下给数据处理器发出命令信号，同时接收来自处理器的状态变量，确定下一个计算步骤，以确保算法按正确的次序实现。所以，控制器决定数据处理器的操作及操作序列。在完成数据处理单元的设计后，应进行控制单元的设计。

12.2.1　控制方式与控制器结构

1. 控制方式

控制功能可集中于一个控制器，也可以分散于各数据处理单元内部，或者是两者的组合。

所以控制方式有三种类型：集中控制、分散控制和集散控制。

数字系统中，如果仅有一个控制器，由它控制整个系统算法的执行，被称为集中控制型。这种类型集中管理各个子运算（子系统）执行的顺序。控制器发出控制信号，使一个或多个处理器进行工作，同时接收各个处理器馈送来的状态变量信息，以便确定后续的控制信号。

集中控制方式经常有一个同步时钟信号。在统一的时钟信号作用下，集中控制和管理各个处理器。每个处理器的操作在时钟信号的节拍下顺序工作。微型计算机就是一个集中控制型的数字系统例子。

在分散控制型的系统中没有统一的控制器，全部控制功能分散在各个子系统中完成。分散控制的时序可以是同步的，也可以是异步的。各子系统之间的输入/输出信号及系统信号相互关联。各子系统可以同时工作，也可以在关联的控制信号作用下顺序地进行工作。

在工业控制系统中，大多采用集散型控制系统。它是集中管理，分散控制。系统中配有系统控制器即中央控制器，集中控制和协调管理各子系统之间总的执行顺序和步骤。但各子系统有自己的控制器，子系统在自己的控制器控制下进行工作。因此集散型控制系统具有集中控制型和分散控制型的特点，在一个复杂的控制系统中得到广泛应用。

2. 控制器结构

① 控制器模型如图 12.2.1 所示。控制器决定算法步骤，控制数据处理器的操作序列必须有记忆能力，应包含存储器（或寄存器）。存储器记忆控制器处在哪一个计算步骤即控制器的状态。在一个状态下，控制器根据接收到处理器反馈的状态变量和外部控制信号产生对处理器的控制信号 T 和输出信号。在下一时钟到来时，控制器转换到下一个状态。显然，控制器模型结构与同步时序电路是一致的。

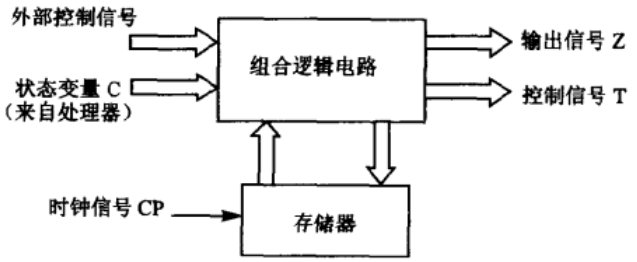


图 12.2.1 控制器模型结构

② 系统同步。其同步是指控制器与外部控制信号和来自处理器的反馈状态变量之间的同步，同时也是指系统控制器向外部输出的同步。为了保证系统正常工作，应将异步信号转换成同步信号。这里介绍两种实现异步信号同步化的电路。

图 12.2.2(a)所示电路由两个 D 触发器组成，实现异步输入信号有效持续时间较长的同步化问题即电平同步；图 12.2.3 所示电路由一个基本 RS 触发器和 D 触发器组成，实现异步输入信号有效持续时间短暂的同步化问题即脉冲同步。这两个电路的同步化时间都发生在时钟的上升沿，也可以发生在时钟的下降沿。在系统中同步化和控制器状态变化可以分别发生在一个时钟脉冲的上、下跳沿（或下、上跳沿），也可发生在连续两个时钟脉冲的对应跳变沿。这可由设计者决定。

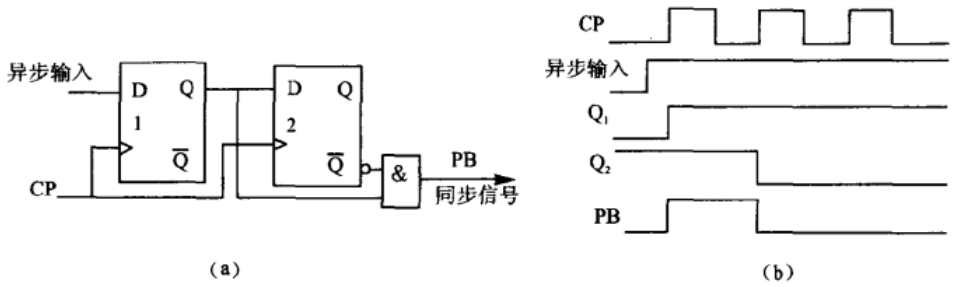


图 12.2.2　第一种异步信号同步化电路

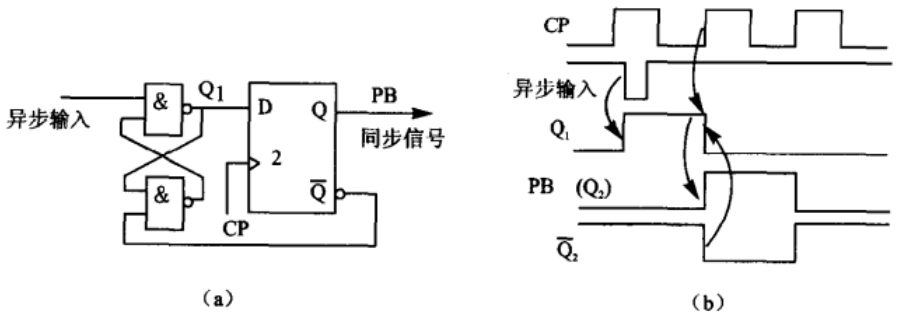


图 12.2.3　第二种异步信号同步化电路

12.2.2　控制单元的实现方法

由前面讨论可知，系统的控制单元本质上就是同步时序电路，因此同步时序电路的设计方法完全适用于控制单元的硬件设计，两者的差别主要表现在两方面。一方面，同步时序电路的设计是根据状态转换图(表)，而控制单元的设计是根据算法流程图和 ASM 图及其他描述形式；另一方面，控制单元的设计是在反复优化算法结构，并且已经完成数据处理单元设计后进行的，一般不需要再进行状态化简。其设计步骤如下：

① 根据系统设计要求，建立描述控制器工作过程的算法流程图和 ASM 图。明确系统的工作状态、判别分支、状态输出和条件输出。也可直接由算法流程图、硬件描述语言编写的程序来设计控制器。

② 选择控制器的硬件类型，包括状态寄存器(存储器)的类型及新状态激励电路和输出电路类型。

③ 状态分配，与同步时序电路的状态分配原则相似。

④ 导出激励函数和输出函数。

⑤ 画出逻辑电路图。

1. 以触发器为核心的控制器设计

例 12.2.1　某系统控制器的 ASM 图，如图 12.2.4 所示。试导出该系统控制器的逻辑图。

① 对 ASM 图进行状态分配，参照时序电路状态分配的原则。由于共有五个状态，需用三位状态变量 Q_2，Q_1，Q_0 编码：$S_0=000$，$S_1=001$，$S_2=011$，$S_3=010$，$S_4=110$ 。

② 填写激励表，以选择D触发器作为控制器的状态寄存器，记忆控制器的工作状态。因此填写三个D触发器激励函数卡诺图是设计中的重要步骤。

首先将ASM图转换成表12.2.1所列的状态表，表中每一行均表示某一个现态和某种输入情况下电路将达到的次态、为达到这一次态所需的激励以及电路的输出。然后填写激励函数卡诺图，如图12.2.5所示，在卡诺图的小方格中填入各自次态的编码值或输入信号变量（即外部控制输入和处理器反馈的状态变量）。

由卡诺图求得激励函数为

$$D_2=\overline{Q}_2Q_1\overline{Q}_0$$

$$D_1=Q_0+\overline{Q}_2Q_1+Q_2K$$

$$D_0=\overline{Q}_1\overline{Q}_0ST+\overline{Q}_1Q_0+Q_2K$$

表12.2.1 由ASM图得到的控制器状态表

状态符号	现态			输入		次态			激励			输出					
	Q_2	Q_1	Q_0	ST	K	Q_2	Q_1	Q_0	D_2	D_1	D_0	Z_1	Z_2	Z_3	Z_4	Z_5	$\overline{CR}$
S_0	0	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0	0
S_0	0	0	0	1	X	0	0	1	0	0	1	0	0	0	0	0	0
S_1	0	0	1	X	X	0	1	1	0	1	1	1	0	0	0	0	1
S_2	0	1	1	X	X	0	1	0	0	1	0	0	0	1	0	0	1
S_3	0	1	0	X	X	1	1	0	1	1	0	0	0	0	1	0	1
S_4	1	1	0	X	0	0	0	0	0	0	0	0	1	0	0	0	1
S_4	1	1	0	X	1	0	1	1	0	1	1	0	1	0	0	1	1

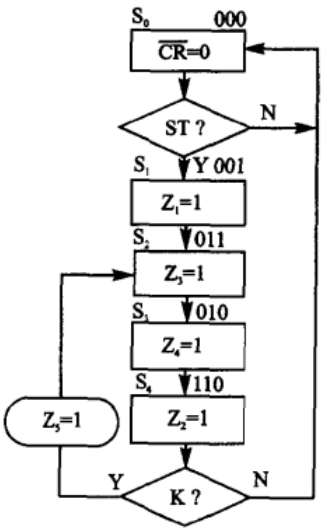


图12.2.4 ASM图

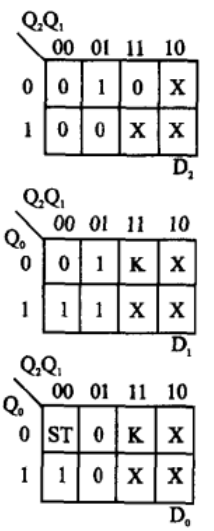


图12.2.5 激励函数卡诺图

③ 输出函数方程，在ASM图中每个状态框中表明了该状态的输出，条件输出（条件框）由椭圆框表示。由ASM图求出输出函数方程十分简单，只是应注意输出信号的极性问题即高

电平有效还是低电平有效。其输出信号方程为

$$\overline{CR} = \overline{S}_0 = \overline{\overline{Q}_2\overline{Q}_1\overline{Q}_0}$$

$$Z_1 = S_1 = \overline{Q}_2\overline{Q}_1Q_0$$

$$Z_2 = S_4 = Q_2Q_1\overline{Q}_0$$

$$Z_3 = S_2 = \overline{Q}_2Q_1Q_0$$

$$Z_4 = S_3 = \overline{Q}_2Q_1\overline{Q}_0$$

$$Z_5 = S_4K = Q_2Q_1\overline{Q}_0K$$

④ 逻辑电路，本例控制器逻辑电路图如图 12.2.6 所示。图中 ST 是系统外部的输入，K 是数据处理单元送来的状态变量反馈信号。采用数据选择器构成激励电路，用触发器记忆控制器状态，用译码器实现输出电路。

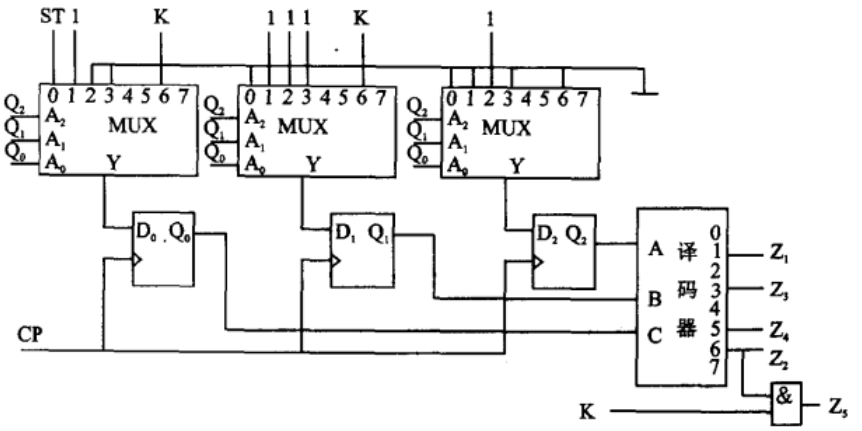


图 12.2.6　控制器逻辑电路图

这种“选择器＋寄存器＋译码器”的控制器硬件结构与 ASM 图的控制算法有较明显的对应关系。如果控制算法有所变化，那么只需更改数据选择器有关输入端，不必改变激励电路硬件。但该结构是以增加硬件成本为代价换取与算法的对应，如果控制器的状态增加（即状态位数增加），那么作为激励电路的数据选择器的通道数就要急剧增加，实现时就很困难。

2. 以计数器为核心的控制器设计

由于模为 M 的计数器具有 M 个状态，因此可用作状态数小于或等于 M 的控制器的状态寄存器。这时控制器的状态转换可利用计数器的计数操作（加 1 或减 1 运算）来实现，而其他状态转移分支又可用计数器的各种逻辑操作（计数、置数、保持等）来完成。在设计中应注意以下两点：

① 根据 ASM 图进行状态分配，其状态分配原则是次态编码尽可能为现态代码加 1（或减 1）；对于状态有多个转移分支的次态的编码，可利用计数、并行置数或保持等逻辑操作来区分不同代码。

② 利用计数器操作图，求得计数器各功能控制端和并行置数端的激励函数。

例 12.2.2　某系统控制器的 ASM 图如图 12.2.7 所示，求出以计数器为核心的该控制器的逻辑图。

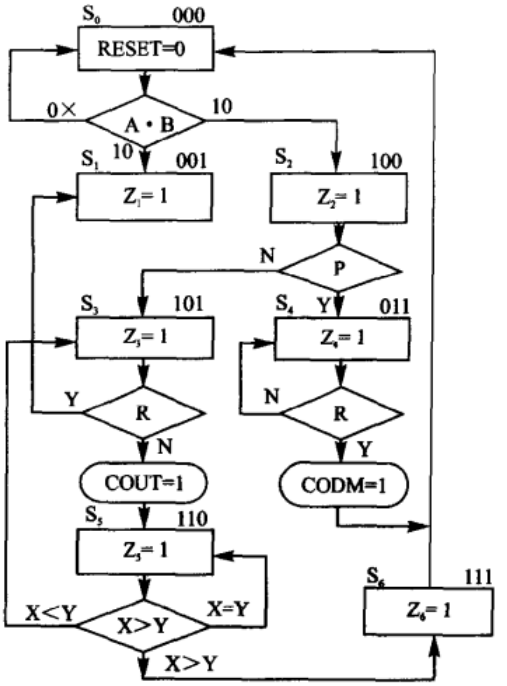


图 12.2.7 ASM 图

① 选择计数器。根据本例 ASM 图中的状态数(7 个状态),选择的计数器模 M≥7,且具有计数和并行置数功能的同步计数器。假设计数器的逻辑功能表如表 12.2.2 所列,逻辑图如图 12.2.8(a)所示。

表 12.2.2 计数器的功能表

输入							输出		
C_r	CE	LD	CP	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	×	×	×	×	×	×	0	0	0
1	1	0	↑	d_2	d_1	d_0	d_2	d_1	d_0
1	0	×	×	×	×	×	保持		
1	1	1	↑	×	×	×	计数(加 1)		

② 状态分配结果如图 12.2.8(b)所示。

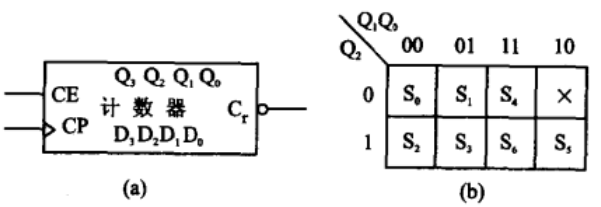


图 12.2.8 计数器框图和状态分配

③ 利用计数器操作图，求得计数器各功能控制端和并行置数端的激励函数。首先画出计数器操作图，如图 12.2.9(a)所示。该图每一方格表示 ASM 图中的某状态为实现状态转换时，计数器所需进行的全部操作。观察在 ASM 图中状态 S_0 的分支情况，当 AB=0× 时，$S_0 \rightarrow S_0$（保持）；当 AB=11 时，$S_0 \rightarrow S_2$（000→100 置数）；当 AB=10 时，$S_0 \rightarrow S_1$（000→001 计数）。

按照状态分配和计数器操作图填写计数器各个控制端(CE,LD)和数据输入端(D)的卡诺图，如图 12.2.9(b)所示。在数据输入端($D_2D_1D_0$)的卡诺图中，每方格填入的次状态是需要并行置数的状态代码，如状态 S_0 的次态有三个 S_0，S_1，S_2，则只有 $S_0 \rightarrow S_2$（000→100 置数）时，需要计数器实现并行置数操作，因此在卡诺图中只填写 S_0 的次态 S_2 的状态代码(100)。计数器各个控制端(CE,LD)和数据输入端(D)的激励方程为

$$C=1$$

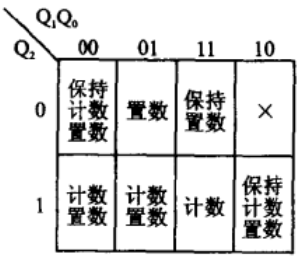
$$CE=m_0A+m_1+m_3R+m_4+m_5+m_6(X=Y)+m_7$$

$$LD=m_0\overline{B}+m_4\overline{P}++m_5\overline{R}+m_6(X>Y)+m_7$$

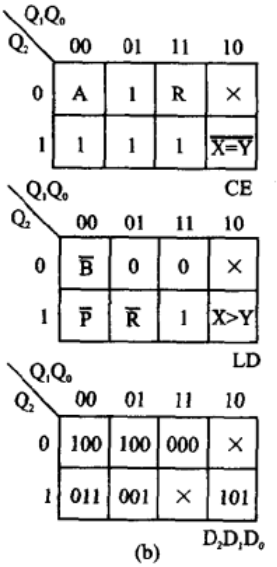
$$D_2=m_0+m_1+m_6$$

$$D_1=m_4$$

$$D_0=m_4+m_5+m_6$$



(a)



(b)

图 12.2.9　计数器操作图和激励函数卡诺图

输出信号方程为

$$RESET=\overline{m}_0\text{（低电平有效）}$$

$$Z_1=m_1,\quad Z_2=m_4,\quad Z_3=m_5,\quad Z_4=m_3,\quad Z_5=m_6,\quad Z_6=m_7$$

$$COUT=m_5\overline{R},\quad CODM=m_3R$$

④ 控制器逻辑图，以计数器为核心，选用“计数器＋数据选择器＋译码器”结构的控制器逻辑图如图 12.2.10 所示。

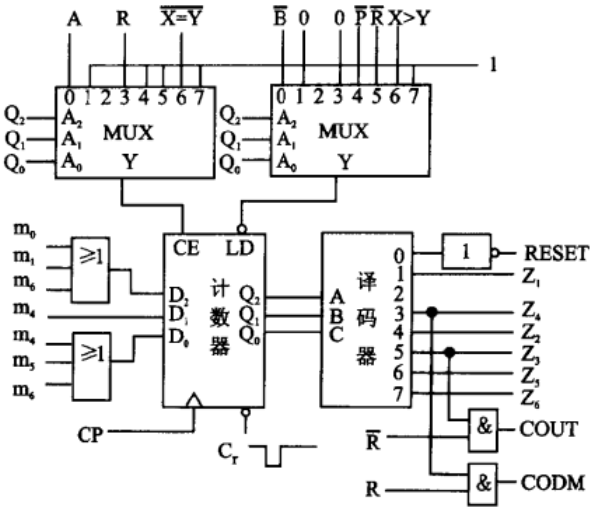


图 12.2.10 控制器逻辑电路图

3. 以移位寄存器为核心的控制器设计

移位寄存器是一种常用的时序电路部件，可用移位寄存器为核心来形成控制器的结构。在这种结构中，移位寄存器用以存储状态，并且利用移位寄存器的各种逻辑功能即左移、右移、保持、并行置数等实现控制器的状态转换，产生控制信号。

例 12.2.3 根据图 12.2.7 所示的 ASM 图，设计以移位寄存器为核心的控制器。

① 选择移位寄存器。由 ASM 图中的状态数 M＝7 选择移位寄存器位数 n≥3（n≥$\log_2$ M）。这里选择常见的四位双向移位寄存器 74194 来设计，其功能表如表 12.2.3 所列。表中 C_r为清零端(低电平有效)；D_0，D_1，D_2，D_3为并行数据输入端；D_{SL}为左移串行数据输入端；D_{SR}为右移串行数据输入端；M_1，M_0为工作方式控制端；Q_0，Q_1，Q_2，Q_3为输出端。

表 12.2.3 移位寄存器 74194 的功能表

输入										输出			
C_r	M_1	M_0	CP	D_{SL}	D_{SR}	D_0	D_1	D_2	D_3	Q_0	Q_1	Q_2	Q_3
0	×	×	×	×	×	×	×	×	×	0	0	0	0
1	0	0	×	×	×	×	×	×	×	保持			
1	0	1	↑	×	d_R	×	×	×	×	右移(Q_0＝d_R)			
1	1	0	↑	d_L	×	×	×	×	×	左移(Q_3＝d_L)			
1	1	1	↑	×	×	d_0	d_1	d_2	d_3	d_0	d_1	d_2	d_3（置入）

② 状态分配。其分配的原则是：次态编码尽可能由现态代码的左移(或右移)获得。本例对 ASM 图的状态分配如图 12.2.11(a)所示，这状态编码只是多种分配方案中的一种。其状态仅用 Q_0，Q_1，Q_2三位状态变量，采用右移操作实现状态转换。

③ 列出移位寄存器的操作图，如图 12.2.11(b)所示。它由已确定的状态代码和 ASM 图中的状态转换情况，来填写移位寄存器所需要进行的操作(保持、左或右移、置数)。为了简化设计和电路结构，尽量只用左移或右移操作中的一种。

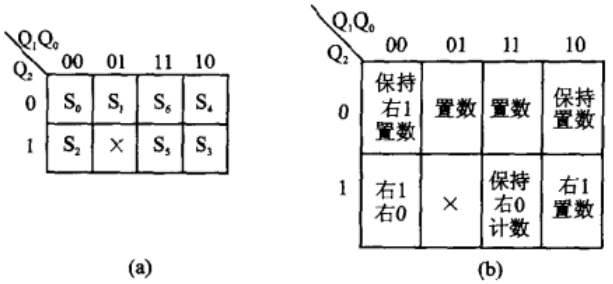


图 12.2.11　状态分配和移位寄存器的操作图

④ 画出移位寄存器各功能控制端和数据输入端(串入、并入)的激励函数卡诺图。本例无左移操作，$D_{SL}=\times$。工作方式控制端 M_1M_0、右移串行数据输入端 D_{SR} 和并行数据输入端 D_0，D_1，D_2 的卡诺图如图 12.2.12 所示。

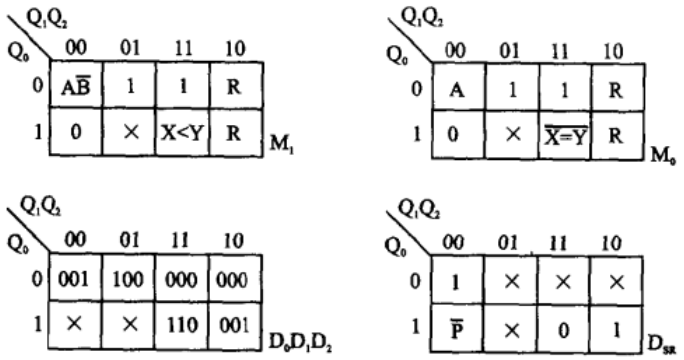


图 12.2.12　移位寄存器激励函数卡诺图

⑤ 控制器逻辑电路图如图 12.2.13 所示。采用“移位寄存器＋数据选择器＋译码器”结构的控制器。

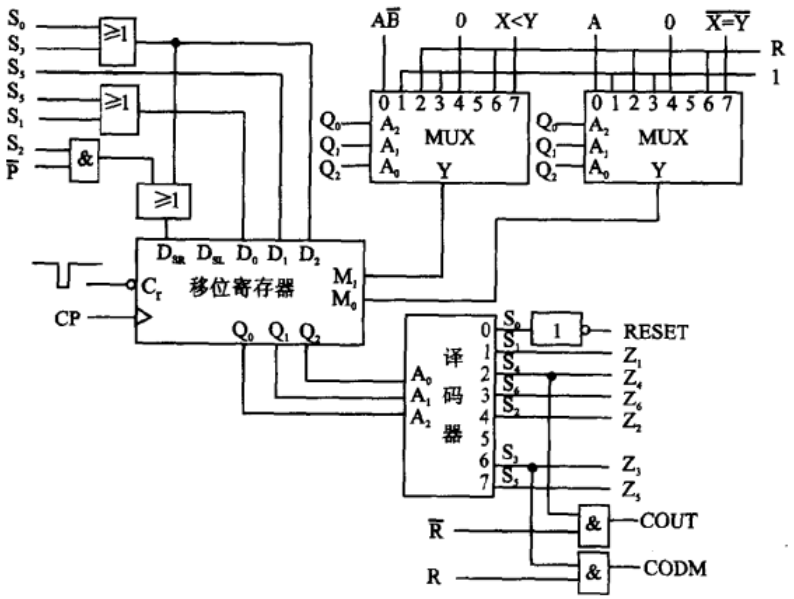


图 12.2.13　控制器逻辑电路图

12.3 数字系统设计实例

前面已介绍了可编程逻辑器件的结构、工作原理和编程技术，以及系统设计方法。本节通过实例详细地给出在 EDA 开发软件的支持下，采用自顶向下的设计方法，实现可编程逻辑器件的数字系统。

首先根据设计任务，确定系统的结构和算法流程图，然后利用不同输入方式(图形、文本、波形)对每个功能模块进行描述，接着在 EDA 开发软件的帮助下，进行设计实现、设计验证和器件编程，完成基于可编程逻辑器件的系统设计。

12.3.1 十字路口交通信号的控制系统

1. 系统的功能要求

十字路口的示意图如图 12.3.1 所示。该系统的功能要求：在十字路口有主干道(A 方向)和支干道(B 方向)，安装有红、黄、绿三色信号灯，指挥车辆安全高效地通行。Ca，Cb 分别为 A，B 方向检测车辆的传感器输出信号。当公路有车时传感器输出为高电平(Ca＝1，Cb＝1)。在初始情况下，A 方向绿灯亮，B 方向红灯亮。当 A，B 两方向只有一方向有车并请求通行时(即传感器为高时)，该方向通行；在其他情况下 A，B 两方向的车辆轮流通行。主干道 A 方向的通行时间(Ga 绿灯亮)为 t_1(50 s)，支干道 B 方向的通行时间(Gb 绿灯亮)为 t_2(30 s)，另外黄灯亮的时间为 t_3(4 s)。

该系统的结构图如图 12.3.2 所示，由控制器和三个受控制的定时器组成。三个定时器分别确定主干道、支干道通行时间以及公共停车(黄灯亮)时间。Z_1，Z_2，Z_3 分别为这些定时器的工作使能信号，当 Z_1，Z_2，Z_3 为 1 时，对应的定时器计数；C_1，C_2，C_3 为每个定时器的状态输出信号。当定时器计数结束时，这些输出信号为 1 。输入信号有 reset 复位信号、clk 秒时钟信号，Ca 和 Cb 为传感器的输入信号。Ra，Ya，Ga 和 Rb，Yb，Gb 分别为 A 方向和 B 方向红、黄、绿灯的输出信号。

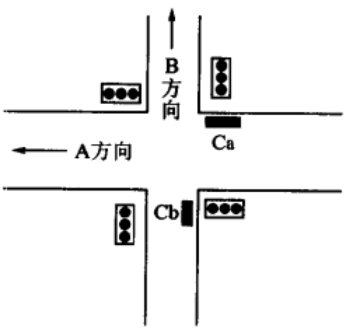


图 12.3.1 十字路口示意图

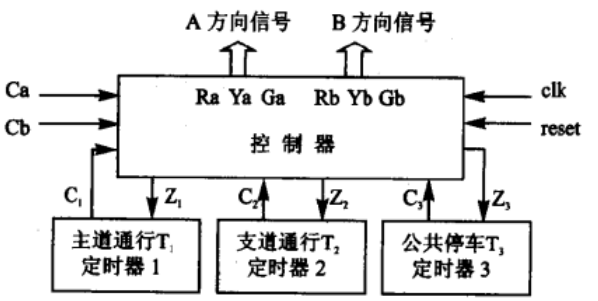


图 12.3.2 交通信号控制系统的结构图

2. 系统算法设计

根据该系统设计要求和结构图，列出控制系统的算法流程图(或 ASM 图)，如图 12.3.3 所示。

3. 设计输入

根据该系统结构图和算法流程图，在 EDA 开发软件的支持下，可以进行可编程器件的设计。先要选择设计输入方式，本设计采用分层描述，用图形输入方式来描述交通信号控制系统的顶层文件，如图 12.3.4 所示。它用框图形式表明了控制器模块(control_jt)和三个定时计数器模块(time1_50，time2_30，time3_4)之间的逻辑关系。每个模块的功能(底层文件)用文本输入方式进行描述。以下为各模块的 AHDL 源文件描述。

(1) 控制器模块(control_jt)

```
SUBDESIGN control_jt
(
    clk , reset          : INPUT ;
    ca,cb,c1,c2,c3       : INPUT ;
    ra,ya,ga,rb,yb,gb    : OUTPUT;
    z1,z2,z3             : OUTPUT;
)
VARIABLE
    ss: MACHINE WITH STATES ( s0, s1,
s2,s3 );
BEGIN
    ss.clk=clk;
    ss.reset=reset; -- 复位端 reset(高电平有效)
 IF reset THEN
    ss=s0;
```

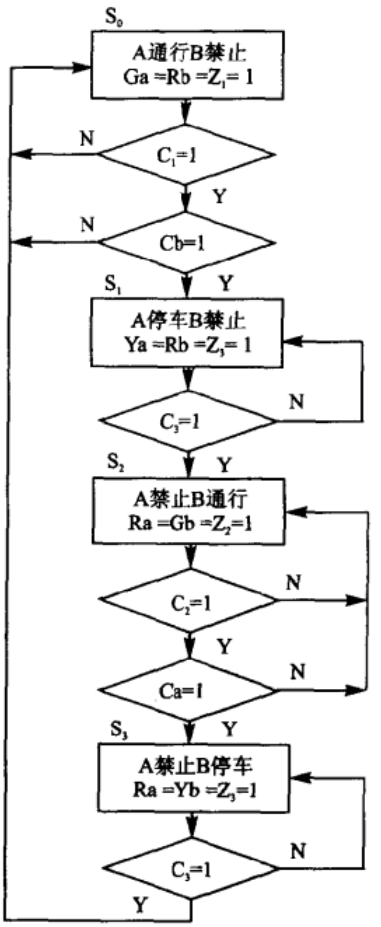


图 12.3.3　交通信号控制系统的算法流程图

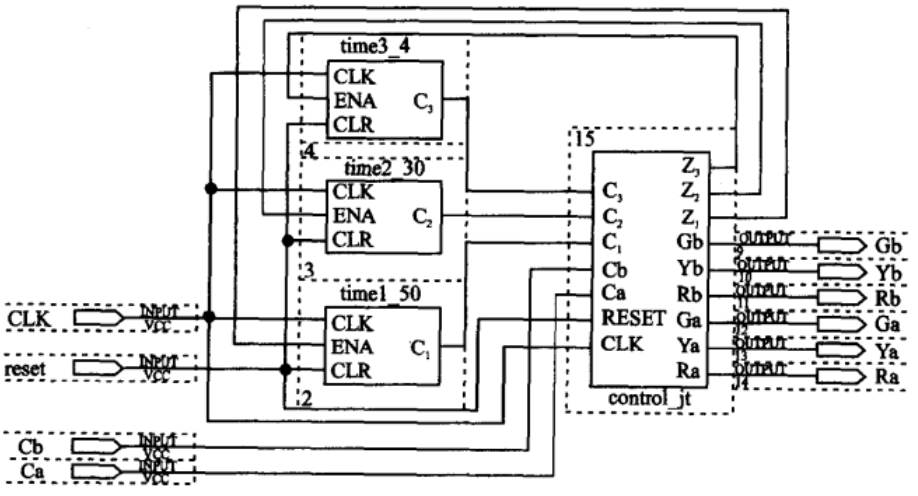


图 12.3.4　交通信号控制系统的原理图(顶层文件)

```
    ELSE
      CASE ss IS
      WHEN s0=>                                  -- A方向通行,B方向禁示
            ga=VCC;
            rb=VCC;
            z1=VCC;
            IF (c1&cb) THEN ss=s1; END IF;
      WHEN s1=>                                  -- A方向停车,B方向禁示
         ya=VCC;
            rb=VCC;
            z3=VCC;
            IF C3 THEN ss=s2; END IF;
      WHEN s2 =>                                 -- A方向禁示,B方向通行
        ra=VCC;
            gb=VCC;
            z2=VCC;
            IF (c2&ca) THEN ss=s3; END IF;
      WHEN s3=>                                  -- A方向禁示,B方向停车
        ra=VCC;
            yb=VCC;
            z3=VCC;
            IF C3 THEN ss=s0; END IF;
        END CASE;
    END IF;
END;
```

(2) 定时器模块

```
    SUBDESIGN time1_50                           -- 定时器1
(
   clk, ena, clr              : INPUT;
   c1                         : OUTPUT;
)
VARIABLE
   count[6..0] ,s             : DFFE;
BEGIN
   count[].clk=clk;
   count[].ena=ena;
   count[].clrn=!clr;
   s.clk=clk;
   s.clrn=!clr;
   c1=s.q;
  IF(count[].q<H"49") THEN                       -- 五十进制计数器
-- IF(count[3..0].q==9) THEN                     % 二进制数转换为BCD码 %
```

```
--          count[].d=count[].q+7;
--    ELSE
      count[].d=count[].q+1;
      s.d=GND;
  --  END IF;
  ELSE
    count[].d=0;
    s.d=VCC;
  END IF;
END;

   SUBDESIGN time2_30                          -- 定时器 2
(
   clk,ena,clr              : INPUT;
   c2                       : OUTPUT;
)
VARIABLE
   count[4..0],S            : DFFE;
BEGIN
   count[].clk=clk;
   count[].ena=ena;
   count[].clrn=!clr;
   s.clk=clk;
   s.clrn=!clr;
   c2=s.q;
  IF(count[].q<29) THEN                        -- 三十进制计数器
     count[].d=count[].q+1;
     s.d=GND;
  ELSE
     count[].d=0;
     s.d=VCC;
  END IF;
END;

    SUBDESIGN time3_4                          -- 定时器 3
(
   clk, ena, clr            : INPUT;
   c3                       : OUTPUT;
)
VARIABLE
   count[1..0],s            : DFFE;            -- 2 位二进制计数器
BEGIN
    count[].clk=clk;
```

```
    count[].ena=ena;
    count[].clrn=!clr;
    s.clk=clk;
    s.clrn=!clr;
    c3=s.q;
    count[].d=count[].q+1;
  IF(count[].q==3) THEN
    s.d=VCC;
  END IF;
END;
```

对图 12.3.4 的原理图和各功能模块的源文件经过编译和仿真，确定正确无误后，可由 EDA 开发软件生成该系统的目标文件，然后下载数据，编程器件。

12.3.2 FIR 数字滤波器

本节摘自《FPGA 原理、设计与应用》。

数字滤波器是数字信号处理 DSP 技术的一个重要分支，利用它可以在形形色色的信号中提取需要的信号和抑制不需要的信号(干扰、噪声)。数字滤波器实质上是用一有限精度算法实现离散时间线性非时变系统，以完成对信号进行滤波处理的功能。数字滤波器对模拟信号进行处理的过程如图 12.3.5 所示。在图中输入信号 $x_i(t)$ 经过低通滤波器 $H_1(S)$ 预处理后，进行抽样、量化(即所谓的 A/D 转换)，得到数字滤波器 H(z)滤波的输入信号 x(n)。它是一组由模拟信号经过取样和量化的数字量，数字滤波器的输出信号 y(n)经过 D/A 转换，再通过另一个低通滤波器 $H_2(s)$，以便将 D/A 输出的模拟量良好地恢复成时间连续信号。

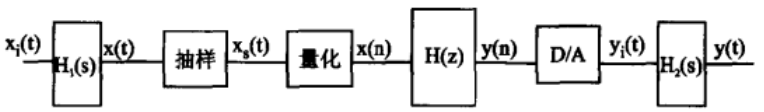


图 12.3.5 数字滤波器处理模拟信号过程框图

1. FIR 结构和设计算法

数字滤波器根据单位冲激响应 h(n)的时间特性分为无限冲激响应(IIR)数字滤波器和有限冲激响应(FIR)数字滤波器两种。有限冲激响应(FIR)数字滤波器按照基本结构分为直接型、级联型和频率抽样型三种。本节重点介绍采用 FELEX10K 系列芯片进行有限冲激响应直接型数字滤波器设计。

有限冲激响应数字滤波器直接型的结构如图 12.3.6 所示。它是用一条均匀间隔抽头的延迟线对抽头信号进行加权求和而构成的。

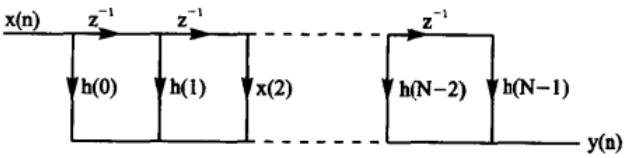


图 12.3.6 FIR 数字滤波器直接形式

上述结构的 FIR 数字滤波器的输入/输出关系所用的时域卷积定理为

$$y(n) = \sum_{m=0}^{N-1} h(m)x(n-m) \qquad (12-1)$$

根据式(12－1),可以看出数字滤波器涉及到大量的卷积运算,使用硬件实现时会占用大量的资源。为了避免这种情况出现,可充分利用 FELEX10K 系列芯片具有的查找表结构,将卷积运算转化为查表移位求和实现。下面以一个简单的卷积运算为例说明硬件实现结构。

对于式

$$y = x(1)h(1) + x(2)h(2) + x(3)h(3) + x(4)h(4) \qquad (12-2)$$

假设 x 和 h 都是无符号整型二进制数,宽度为两位(2 bit),取值如下:

$$h(1) = 01,\quad h(2) = 11,\quad h(3) = 10,\quad h(4) = 11$$

$$x(1) = 11,\quad x(2) = 00,\quad x(3) = 10,\quad x(4) = 01$$

图 12.3.7 表示式(12－2)运算的实现。在图中可以看到运算纵向和横向分别实现。中间数据 P1(n)中的四个数据(每 2 bit)实际上是乘数 x(n)的最低位比特与 h(n)相乘的结果,并且该值不是 0 就是 h(n)。这是因为二进制的取值只有 0 和 1。进一步看,中间数据 P1 和 P2 的值,即"100","011"由不同的 h(n)之和构成,而对 h(n)的选择是由乘数 x(n)的相同位的比特(高位比特或低位比特)决定的。如图 12.3.7 中每 x(n)的低位比特组成为 1001,则 P1 的值为h(1)＋h(4);每 x(n)的高位比特组成为 1010,则 P2 的值为 h(1)＋h(3)。因此利用 Altera 公司 FLEX 器件中的查找表(LUT)结构,预先将 h(n)的各种组合(在本例中从 0000～1111 共 16 种)存入查找表,则上例中的原需 4 次乘法和 3 次加法的卷积运算转化为 1 次加法。图 12.3.8 显示了用查找表实现该例的结构。

被乘数 h (n) →	01	11	10	11	
乘数 x (n) →	11	00	10	01	
中间数据 P1 (n) →	01	00	00	11	= 100
中间数据 P2 (n) →	+ 01	00	10	00	=011
	011	000	100	011	=1010

图 12.3.7　卷积运算过程

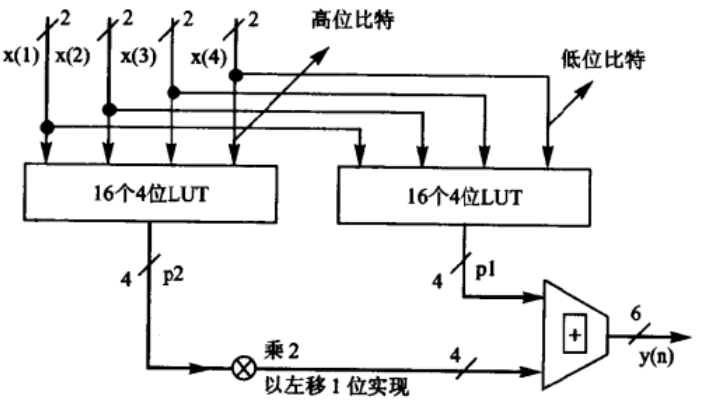


图 12.3.8　查找表实现卷积运算

用查找表实现卷积运算时，有并行和串行两种结构。图 12.3.8 显示的就是并行结构，其中的两个 LUT 是完全相同的。在并行结构中，LUT 的数量由 x(n)的数据宽度决定。一位对应一个 LUT，这样速度能够达到最大，但占用资源也相当可观。而串行结构中，只需一个 LUT，x(n)的每位比特串行查表，并利用累加器累加得出最后结果。显然串行结构比并行结构占用资源要少得多，但代价是处理速度降低。

为了充分说明 FELEX10K 系列芯片查找表结构的使用方法，这里对 FIR 数字滤波器的设计采取全局并行的方案，即将输入 x(n)经过不同的延时后同时进行处理。FIR 数字滤波器可分为五个功能模块(控制器、延时环节、并/串转换、抽头系数、移位相加)，其结构图如图 12.3.9 所示。

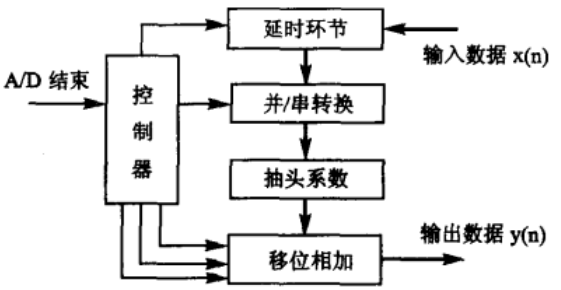


图 12.3.9 FIR 数字滤波器结构图

2. 设计输入

根据图 12.3.9 所示的 FIR 数字滤波器结构图，写出各功能模块的 AHDL 源文件如下。

(1) 延时环节模块

延时环节模块(filter shift)使 A/D 转换后的数据通过不同的触发器，产生不同的延时。该模块采用文本编辑方式进行设计，源程序如下：

```
PARAMETERS
(
        tap=3,
       x_wid=12,
      );
    SUBDESIGN filter_shift
(
     clk                        : INPUT;
     clr                        : INPUT=VCC;
     xin[x_Wid..1]              : INPUT;
     s_en                       : INPUT=GND;     -- 移位使能端
     y_ff[tap..1][x_wid..1]     : OUTPUT;
)
VARIABLE
    shifter_ff[tap..1][x_wid..1]   : DFF
BEGIN                                            -- 连接移位寄存器的输入端
     shifter_ff[ ][ ].clrn =clr;
```

```
        shifter_ff[ ][ ].clk=clk;
        shifter_ff[ ][ ].ena=s_en;
        shifter_ff[ ][ ]. d=xin[];
        FOR i IN 2 TO tap GENERATE
            Shifter_ff [ i][ 1] =shifte_ff[i-1][].q;
        END GENERATE;
        FOR i IN 1 TO tap GENERATE
            y_ff[ ][ ]=shifter_ff[ i][ ].q;
        END GENERATE;
    END;
```

(2) 并/串转换模块

并/串转换模块将通过延时模块产生的不同延时分别同时转换为查找表的串行地址，提供给抽头系数模块。并/串转换模块包含一个单通道并/串转换子模块(p_s 模块)。这里先介绍 p_s 子模块，然后再介绍并/串转换模块。

① 单通道并/串转换子模块(p_s 模块)。

该模块能够将一个通道的并行数据转换为串行数据。程序中，将并行数据宽度参数化，以便能够适应不同宽度数据的需要。模块采用文本编辑方式进行设计，源程序如下：

```
    PARAMETERS
(
    width=12                                    -- 并入宽度
);
SUBDESIGN p_s
(
    paralle_in[width..1]        : INPUT;        -- 并入
    serial_out                  : OUTPUT;       -- 串入
    clk                         : INPUT;
    clr                         : INPUT=VCC;
    Load_n                      : INPUT=GND;
)
VARIABLE
    reg[width..1]               : DEF;
BEGIN                                           -- 给所有的触发器连接时钟和清除信号
    Reg[].clk=clk;
    Reg[].clm=clr;
  IF(load_n)THEN                                -- 装载
    FOR j in   1 To width GENERATE
        reg[j].d=parallel_in[j];
    END GENERATE;
 ·ELSE
    FOR j   in 1 TO width-1 GENERATE            -- 移位
        reg[j].d=reg[j+1].q;
    END GENERATE;
```

```
        reg[width]. d=GND;
      END IF;
        serial_out=reg[1]. q;
    END;
```

② 并/串转换模块(s_term 模块)。在单通道并/串转换的基础上,并/串转换模块实现多通道数据的并/串转换。模块采用文本编辑方式进行设计,源程序如下:

```
    INCLUDE"p_s;
        PARAMETRERS
    (
        in_width= 12 ,                          --进行转换的并行数据宽度
        term_wid                                --进行转换的并行数据通道数
    );
        SUBDESIGN s_term
    (
        clk                                     ; INPUT;
        clr                                     ; INPUT=VCC;
        data_in[term_wid. . 1][in_widh. . 1]    ; INPUT;
        load                                    ; INPUT=GND;
        rom_ad[term_wid. . 1]                   ; OUTPUT;
        )
    BEGIN
        regs[]. clk=clk;
        regs[]. clr=clr;
        regs[]. load=load;
        FOR i IN 1 TO term wid GENERATE
            regs[i]. parallel_in[]=data_in[][];
        END GENERATE;
        rom_ad[]=regs[]. seial_out;
    END;
```

(3) 抽头系数模块

抽头系数模块(filter_coef)将抽头系数的各种组合固化在 ROM 中。它的地址输入端接收并/串转换模块的串行输出,然后查表得到卷积的中间数据,并将这些数据输出到移位相加模块。模块采用文本编辑方式进行设计,源程序如下:

```
        INCLUDE"lpm_rom";
        PARAMETERS
    (
        addr_wid=3,
        coef_wid= 16,
        ini_file  ="cpef. mif"
        );
    SUBDESIGN filter_cofe
    (
```

```
    clk                                 ;INPUT;
    rom_addr[addr_wid..1]               ;INPUT;
    dataout[coef_wid..1]                ;OUTPUT;
)
VARLABLE
    roml        ;lpm_rom
                WITH( LPM_WIDT=coef_Wid,
                      LPM_WIDTHAD=addr_wid,
                      LPM_FILE=ini_file,
                      LPM_OUTDATA="REGISTERED");
BEGIN
    roml.inclock=clk;
    roml.outclock=clk
    roml.address[]=rom_addr[]
    dataout[]=roml.q[] ;
END;
```

这里需要特别解释的是 LPM_ROM 的初始化文件(coef.mif)以 ASCll 字符表示,后缀为.mif 的 MIF(Memory Initialization File)文件在编译和仿真时说明 RAM 或 ROM 的初始内容。对于 RAM 或 ROM 的每一个地址,MIF 文件包含着相应的初始值。在设计中一般将前面提到的查找表写入到相应 ROM 使用的 MIF 文件中。在编写 MIF 文件时需要说明存储数据的长度和宽度。在下面的例子中,假设抽头系数数据宽度为 16 位,精度为 15 位。其中:

h(1)=0.125(1000H)

h(2)=0.5(2000H)

h(3)=−0.125(A000H)

则相应的 MIF 文件(coef.mif)的编写如下所示:

```
WIDTH=16;                   --说明存储数据宽度
DEPT=8;                     --说明存储数据长度
ADDRESS_RADIX=HEX;          --说明存储数据地址采用十六进制表示
DATA_RADIX=HEX;             --说明存储数据数值采用十六进制表示
CONTENT BEGIN
    0:0000;
    1:1000;                 --(h(1))
    2:2000;                 --(h(2))
    3:3000;                 --(h(1)+h(2))
    4:A000;                 --(h(3))
    5:0000;                 --(h(1)+h(3))
    6:1000;                 --(h(2)+h(3))
    7:2000;                 --(h(1)+h(2)+h(3))
END;
```

在 MIF(coef.mif)文件中,存储数据值和地址时不仅可以使用十六进制(HEX),而且还可以使用二进制(BIN)、八进制(OCT)和十进制(DEC)。在 CONTENT BEGIN 和 END 之间的每一个程序语句中,冒号(:)左边为存储数据地址,右边为存储数据数值;存储数据的数据宽度

与数据个数必须分别与前面为 WIDTH 和 DEPTH 所赋的值相等。当按照上面的格式完成 MIF 文件编写后，在存盘的时候一定要在文件名的后面加上后缀(.mif)。

(4) 移位相加模块

移位相加模块通过将中间数据移位相加而实现两个数相乘的功能。为了充分说明该模块的功能，假设被乘数为 3(0011)、乘数为 13(补码为 1101)，那么乘积过程为

```
      0011
×     1101
------------
      0011
+    0000
+   0011
-  0011
------------
  11110111
```

在上式中，乘数的各位分别为 1 或 0。当为 1 时，被乘数作为中间数据参加移位相加运算；当为 0 时，则由 0 作为中间数据参加移位相加运算。如果乘数数据宽度为 n，则前 n－2 次为加法运算，最后一次为减法运算。因为被乘数、乘数都是以补码参加运算，所以乘积也是以补码的形式出现，这样就涉及到中间数据符号位的问题了。为了说明这个问题，再假设被乘数为－3(补码为 1101)、乘数为 3(0011)。它们的数据宽度都为 4 位，小数点后面的精度为 0；乘积结果数据宽度为 8 位，小数点后面的精度为 0。那么，乘积过程为

```
      1101
×     0011
------------
      1101
     11101        --扩展符号位
+    1101         --第一次进行加法运算
------------
     110111
    1110111       --扩展符号位
+   0000          --第二次进行加法运算
------------
    1110111
   11110111       --扩展符号位
-  0000           --第三次进行减法运算
------------
   11110111
```

在上式中，乘积的结果不仅与符号位有关，而且与被乘数、乘数以及所要求乘积的数据宽度、小数点后的数据精度有关，乘积结果的整数部分可以根据对乘积大小的估计设置为一定宽度。小数部分的宽度可根据精度的要求进行取舍。这样就要求模块对被乘数、乘数和乘积的数据宽度、精度均应设置参数，通过对这些参数赋予不同的值以适应不同的需要。

根据上述的乘积过程设计移位相加模块的源程序如下：

```
INCLUDE "lpm_add_sub";
x_width=16,                 -- 乘数(即要进行累加的数)之宽度:1 位整数、15 位小数
x_pre=15,                   -- 乘数之精度
y_width =12,                -- 被乘数(即决定累加次数的数)之宽度;2 位整数、10 位小数
y_pre=10,                   -- 被乘数之精度
out_int=3,                  -- 乘积的整数部分位数
out_pre=15,                 -- 乘积的小数部分位数
pipline="YES"
    );
```

```
    CONSTANT dff_num=(x_width-x_pre)+(y_width-y_pre)+out_pre;
    CONSTANT out_width= out_int+out_pre;

SUBDESIGN accumulator
(
    clk                         : INPUT;
    clr                         : INPUT=VCC;
    datain[x_width..1]          : INPUT=VCC;
    add_sub                     : INPUT=VCC;        --加减控制
    f_ en                       : INPUT=GND;        --输出使能控制
    A_load                      : INPUT=GND;        --装入数据控制
    dataout[out_width..1]       : OUTPUT;
  VARIABLE
    adder                       : lpm_add_sub
                                  WITH(lpm_width=x_width+1,
                                  lpm_representation="UNSIGNED");
    accum[dff_num..1]           : DFF
    IF (pipeline==“YES”) GENERATE
        Flip[out_width..1]      : DFFE;
    END GENERATE;
    adder_dataa[x_width+1..1]   : NODE;
    adder_datab[x_width+1..1]   : NODE;
    adder_result[x_width+1..1]  : NODE;
BEGIN
    accum[].clk=clk;
    accum[].clm=clr;
    CASE a _load IS
      WHEN VCC=>
            accum[dff_num]=dalain[x_width];
            accum[dff_num-1..dff_num-x_width]=datain
        IF(dff_num-x_width-1> 0) GENERATE
            accum[dff_num-x_width-1..1]=GND;
        END GENERATE;
      WHEN GND=>
        accum[dff_num..dff_num-x-width].d=adder_result[];
        accum[dff_num-x _width-1..1].d=accum[dff_num-x_width..2];
    END CASE;
      adder.add_sub=add_sub;
      adder_datab[]=(datain[x_width],datain[]);        --扩展符号位
      adder_dataa[x_width+1]=accum[dff_num];
      adder_dataa[x_wldth..1]=accum[dff_num-x_width+1];
      adder.dataa[]=adder_dataa[];
      adder.datab[]=adder_datab[];
```

```
        adder_result[]=adder. result[];
      IF( pipeline==“YES”)GENERATE
        flip[]. clk=clk;
        flip[]. clm=clr;
        flip[]. ena=f_en;
        flip[]. d=accum[out_width. . 1]. q;
        dataou[]=flip[]. q;
      ELSE GENERATE
        dataout[]=accum[out_wldth. . 1]. q;
      END GENERATE;
  END;
```

在以上程序的注释中已经对参数的设置进行了解释，而最后需要说明的是乘积结果的数据宽度应该为 28 位。但是，因为整数部分(包括符号位)取 3 位、小数点后面取 15 位已经能满足要求，所以设置乘积结果数据宽度为 18 位。

(5) 控制器模块

上述各种模块虽然各自能够完成一定功能，如延时、并/串转换、移位相加等，但是当它们按一定的形式组合在一起实现滤波器功能时，需要有一系列的控制信号对上述各种模块进行精确的控制。控制器模块(filter_con)在接收到 A/D 转换结束信号(AD_END)后，依次产生移位寄存器使能信号、并行到串行转换的装入信号、移位相加的装入信号、加减控制信号和滤波结果输出信号等各种控制信号，使上述各个模块按照一定的时序进行操作，最终完成滤波功能。该模块采用文本编辑方式进行设计，源程序如下：

```
PARAMETERS
(
    x_ width=   12                    -- 输入数据的宽度，决定 add_sub 和 flip_en 在何时起作用
);
    CONSTANT dff_num=x_width+6;
SUBDESIGN   filter_con
(
    clk             : INPUT;
    clr             : INPUT=VCC;
    ad_end          : INPUT;          -- A/D 转换结束信号
    shifter_en      : OUTPUT;         -- 移位寄存器使能信号
    Ps_load         : OUTPUT;         -- 并行到串行转换的装入信号
    Add_load        : OUTPUT;         -- 移位相加的装入信号
    add_sub         : OUTPUT;         -- 加减控制信号
    flip_en         : OUTPUT;         -- 滤波结果输出信号
)
VARLABLE
      shift[dff_num. . . 1]   : DFFE;
      base                    : NODE;
BEGIN
    shift[]. clk=clk;
```

```
        shift[].clrn=clr;
        shift[1].d=ad_end;
        shift[2].d=shift[1].q;
        base=!shift[1].q&shift[2].q;
        shifter_en=base;      % 以上四句为捕捉 A/D 转换结束信号,并且产生移位寄存器使能信号 %
      shift[3].d= base;
        FOR i IN 4 TO dff_num GENERAT
          shift[i].d=shift[ i-1].q;
        END GENERATE;
          ps_load=shift[3].q;
          add_load=shift[6].q;
          add_sub=!shift[6+x_width-1].q;
         flip_en=shift[6+x_width].q;
    END;                                    -- 以上产生其他控制信号
```

(6) 顶层设计文件

下面介绍数字滤波器的顶层设计文件(Filter)。假设滤波器结构如图 12.3.10 所示。在图中 x(n)的数据宽度为 12 位,其中精度为 10 位;抽头系数(h(1),h(2)和 h(3))的数据宽度为 16 位,其中精度为 15 位;输出结果数据宽度为 18 位,其中精度为 15 位。顶层设计文件采用文本编辑方式设计的源程序如下:

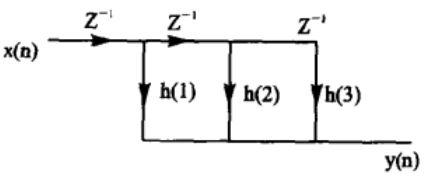


图 12.3.10　FIR 数字滤波器实例结构

```
INCLUDE"filter con";
  INCLUDE"filter shift";
  INCLUDE"p_s";
  INCLUDE"s_term";
  INCLUDE"accumulator"
  INCLUDE"filter_coef";
  PARAMETERS
(
      tap=3,
      x_wid=12,                         -- X 的数据宽度为 12 位
      x_ pre=10,                        -- X 的小数点后有 10 位精度
      coef wid=16,                      -- 滤波器系数数据宽度为 16 位
      coef_pre=15,                      -- 滤波器系数小数点后有 15 位精度
      z_wid=18,                         -- 输出 Z 的数据宽度为 18 位
      z_pre=coef_pre,                   % 输出 Z 精度的小数点后有 15 位精度,2 位整数
                                          位,1 位符号位 %
      pipeline="YES"
  );
  SUBDESIGN filter
  (
      g_clk          : IPUNT;
      clr            : INPUT=VCC;
      ad_end         : INPUT=VCC;       -- A/D 转换完成信号
```

```
    xin[x_wid..1]    : INPUT;              -- 输入数据
    z[z_wid..1]      : OUTPUT;             -- 卷积结果
    )
VARIABLE
    cont             : filter_con WITH(x_width=x_wid,);
    shift_reg        : filter_shift WITH(tap=tap,
                                         x_wid=x_wid,
                                        );
    p_s_c            :s_term   WITH (in_width=x_wid,
                                     term_wid=tap);
    rom_coef         : filter_cofe  WITH(addr_wid=tap,
                                         cofe_wid=coef_wid,
                                         ini_file="coef.mif";
    shift_add        :accumulator WITH(x_width=cofe_wid,        -- 16
                                       x_pre=coef_pre,          -- 15
                                       y_width=x_wid,           -- 12
                                       y_pre=x_pre,             -- 10
                                       pipeline="YES",
                                       out_int=z_wid-z_pre,     -- 3
                                       out_pre=z-pre);          -- 15
    clk:node;
BEGIN
  clk= global(g_clk);

-- 连接控制器模块的输入端
  cont.clk=clk;
  cont.clr=clr;
  cont.ad_end=ad_end;

-- 连接延时环节模块的输入端
  shift_reg.clk=clk;
  shift_reg.clr=clr;
  shift_reg.xin[]=xin
  shift_reg.s_en=cont.shifter

-- 连接并/串转换模块的输入端
  p_s_c.lk =clk;
  p_s_c.clr=clr;
  p_s_c.oad= cont.ps_load;
  p_ s_c.data_in[][]=shift_ reg.y_ff[][];

-- 连接抽头系数模块的输入端
  rom_coef.clk=clk;
```

```
    rom_coef · rom_addr[ ]=p_s_c · rom_ad[ ];

  -- 连接移位相加模块的输入端
    shift_add. clk=clk
    shift _add. clr=clr;
    shift_add. a_load=cont. add_load;
    shift_add. add_sub=cont. add_sub;
    shift_add. f_en= cont. flip_en;
    shift_add. datain[ ]=rom_coef. dataout[ ];
    z[ ]=shift_add. dataout[ ];                    -- 输出卷积结果
  END;
```

为了使读者能直观了解程序，还将顶层设计文件采用图形编辑方式进行设计，如图 12.3.11 所示。

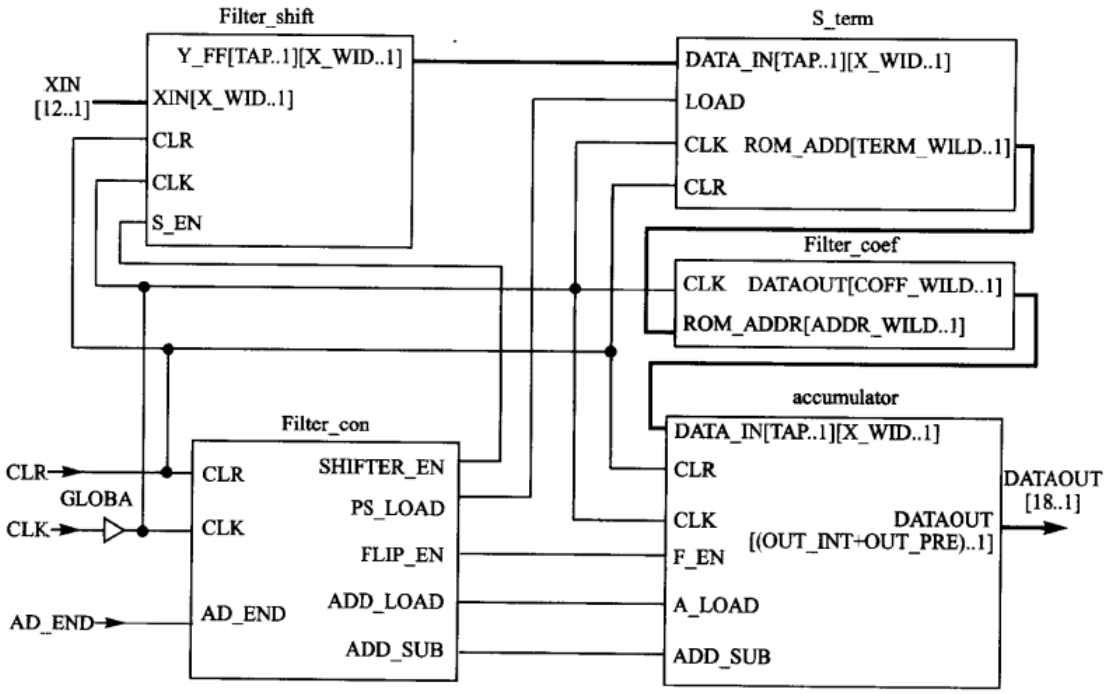


图 12.3.11　数字滤波器原理图(顶层文件)

通过对运用 AHDL 进行数字滤波器设计实例的介绍，着重叙述了在 FLEX10K 芯片中查找表结构的使用方法。因为上例中的抽头系数是固定值，所以存储它们使用 ROM。如果滤波器为自适应，则存储它们要使用 RAM，同时在设计中增加修改抽头系数的模块。但是无论使用 ROM 还是 RAM，都要涉及到查找表。查找表结构不仅能够实现上面提到的卷积运算，而且还可以把许多复杂的数学运算转化为查表。查找表结构使这些数学运算的硬件实现变得十分简单、灵活。另外需要说明的是，在数字滤波器实例中涉及到的子模块如延时模块、并/串转换模块、移位相加模块等都是独立的参数化模块，在其他的应用中根据需要对这些模块的参数进行赋值就可以使用，因而具有很大的灵活性。

附　录

附录一　常用元器件模型参数的使用说明(EWB)

用电子工作台(Electronics Workbench)在进行电路设计的时候,若需要了解元器件特点和使用方法,则可以在选定元器件后,按 F1(在线帮助)键,屏幕会显示该元器件的特点、性能和使用方法等有关信息。下面的说明主要是介绍常用元器件模型的参数定义和使用注意事项。

电池 (Battery):

电压设置必须大于零,而且内阻为零。若将它并联使用,则必须串联 1 mΩ 的电阻。

交流电压源 (AC Voltage Source):

电压数字为均方根(RMS)值。

$$V_{RMS}=V_{peak}/\sqrt{2}$$

交流电流源 (AC Current Source):

电流数字为均方根(RMS)值。

$$I_{RMS}=I_{peak}/\sqrt{2}$$

电压控制电压源 (Voltage Controlled Voltage Source):

$$E=V_{out}/V_{in}$$

电压控制电流源 (Voltage Controlled Current Source):

$$G=I_{out}/V_{in}$$

电流控制电压源 (Current Controlled Voltage Source):

$$H=V_{out}/I_{in}$$

电流控制电流源 (Current Controlled Current Source):

$$F=I_{out}/I_{in}$$

V_{cc}电压源 (Voltage Source):

5 V 电源或逻辑高电平。

V_{dd}电压源 (Voltage Source):

15 V 电源或逻辑高电平。

AM 调幅源 (AM Source):

$$V_{out}=V_c * \sin(2 * (* f_c * t) * (1+m * \sin(2 * (* f_m * t))$$

其中　V_c——载波幅度 V;

f_c——载波频率 Hz;

m——调制指数;

f_m——调制频率 Hz。

FM 调频源 (FM Source)：

$$V_{out} = V_a * \sin(2 * (* f_c * t + m * \sin(2 * (* f_m * t))$$

其中　V_a——峰值幅度 V；

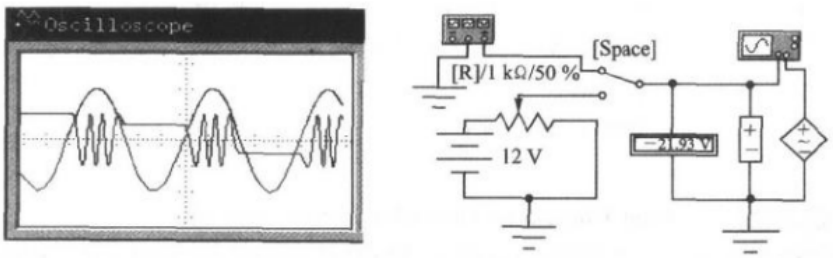
f_c ——载波频率 Hz；

m——调制指数；

f_m——调制频率 Hz。

电压控制正弦波振荡器 (Voltage Controlled Sine Wave Oscillator)：

应用举例：压控振荡器(VCO)电路，如附录图 1.1 所示。输出频率取决于控制电压，该电路的控制电压在 0 V 时，输出频率为 100 Hz；控制电压在 12 V 时，对应的输出频率为20 kHz。控制信号的类型由切换开关进行控制。当控制信号是方波时，输出是频移键控(FSK)信号；当输入信号为正弦波时，输出是调频波信号。



附录图 1.1　电压控制正弦波振荡器

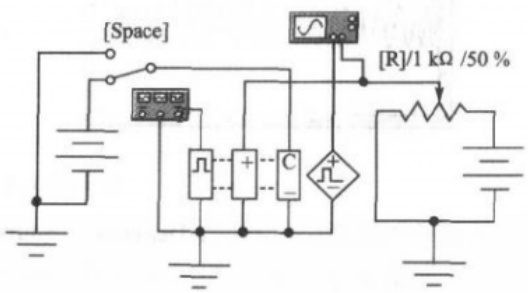
受控单脉冲发生器(Controlled One Shot)：

该器件是一种多用途脉冲波形发生器。输入 AC 和 DC 的电压信号作为分段线性控制曲线中的独立变量(控制量，脉宽)。根据该控制曲线确定振荡器输出的脉冲宽度值。可以调整的时钟触发器数值有：触发器输出延时；脉冲宽度输出延时；输出波形上升和下降时间；以及输出的高、低电平。当在“CLOCK 0”输入端的波形超过预置的门限电平时，输出端就被触发输出高电平。其输出的幅度和上升、下降时间都可以按用途进行设定。脉冲宽度由“CONTROLVOLTAGE(+)”端输入的直流电平或变化的信号进行控制。

应用举例如附录图 1.2 所示。由电位器提供一个可调的直流电压控制输出脉冲宽度，“CLEAR(C)”端在低电平时重新触发，在高电平时将阻止脉冲触发。改变控制电压的波形，可以产生脉宽调制(PWM)波形。为了观察 PWM 效应，将电位器连至“WIDTH ADJ terminal(+)”端。

分段线性源(PWL Source)：

该源可以通过插入不同的时间、电压值控制波形的形状。每一对(时间、电压)数值对应输出曲线该时刻的电压。在两个时间值之间的电压，由线性插值确定。该器件有两个端子如同一个电压源。它读入专门格式的表述时间、电压的文本文件，用这些数据，该元件会产生符合输入文件要求的电压波形。



附录图 1.2　应用举例一

输入文本文件的书写格式:时间<空格>电压。空格的间距没有具体规定。举例:

```
0              0
2.88e-06       0.01876
3.343e-02      0000345
```

若最早的输入点不是在时间零点,则 PWL 源输出的电压是从时间零点(0,0)一直到最早的时间那一点。而对输入最后点以后的电压,PWL 源将维持最后点的电压值,一直保持到仿真结束。PWL 源能够自动处理数据的分类、排列。若没有注明文件名,则 PWL 源相当于短路。

应用举例:下列的文本文件 DATAI. TXT 为方波。

```
0       0          时间起始点 t=0,V=0。
0       1          在 t=0 处,电压跃升 1 V 作为上升沿。
0.1     1          维持 1 V 直到 t=0.1 s。
0.1     0          在 t=0.1 s 处下降到 0 V,作为下降沿。
0.2     0          维持 0 V 直到 t=0.2 s。
0.2     1          上升沿。
0.3     1          维持 1 V 直到 t=0.3 s。
...     ...
```

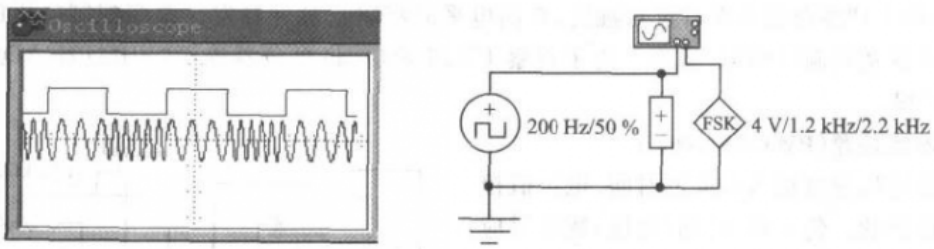
压控分段线性源(Voltage Controlled Piecewise Linear Source):

该源允许使用者插入 5 对数据坐标(输入、输出)控制输出波形的形状。具体数据可以通过参数设置对话框进行了解和设置。若只用二对坐标数据,则输出波形即为线性状态输出。在输入的数据坐标以外,受控 PWL 源还会扩展一定的范围。在输入信号较大的情况下,有可能使输出达到很大或很小值,此时要考虑源的适用能力。为了减少仿真时不收敛的可能性,受控 PWL 源提出了输入平滑范围(Input Smoothing Domain 缩写 ISD)的限制。比如 ISD 设定为10 %,即仿真时,对每个坐标点的平滑半径是小于或等于每个坐标点前后区段的 10 %。

频移键控源(FSK 源):

FSK 源当二进制码"1"送至输入端时,会产生一个传输频率 f1,当输入"0"时,产生空号传输频率 f2。

应用举例如附录图 1.3 所示。



附录图 1.3 应用举例二

非线性相关源(Nonlinear Dependent Source):

可表示的数学关系式和表示式有:+,-,*,/,^,abs,asin, atanh,exp,sin,tan,acos,asinh,cos,ln,sinh,u,acosh,atan,cosh,sqrt,uramp 等函数,也适合表示单位阶跃函数。若相关变量取 V,则输出为电压伏特;若相关变量取 I,则输出为电流安培。

举例：

$$I=\cos(V(1))+\sin(V(2))$$
$$V=\ln(\cos(\log(V(1,2)))-V(3)^{4}+V(2)^{V(1)}$$
$$I=17$$

多项式源(Polynomial Source)：

$$V_{out}=A+B*V_1+C*V_2+D*V_3+E*V_4^2+F*V_1*V_2+G*V_1*V_3+H*V_2^2+I*V_2*V_3+K*V_1*V_2*V_3$$

线性变压器(Linear Transformer)：

变压器特性方程为

$$V=nV_2,\qquad i_1=\frac{1}{n}i_2$$

其中　V_1——初级电压；

V_2——次级电压；

i_1——初级电流；

i_2——次级电流；

n——匝数。

在创建电路和仿真运行时，应将该变压器的初、次级均接地。

继电器(Relay)：

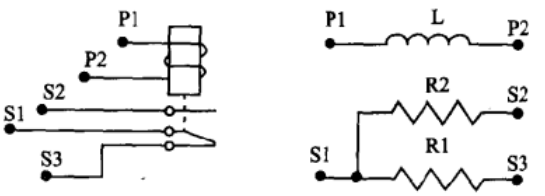
继电器电路见附录图 1.4，继电器特性方程为

R1＝0

R2＝∞　　$I_p \leqslant I_{on}$

R1＝∞

R2＝0　　$I_{hd} < P_{on} \leqslant I_p$



附录图 1.4　继电器特性

其中　L——继电器线圈电感 H；

R——继电器开关接触电阻 Ω；

I_{on}——导通电流 A；

I_{hd}——保持电流 A；

I_p——流过线圈的电流 A。

继电器和开关在并联使用或与电源并联应用时，须串入一个 1 mΩ 的电阻。

延时开关(Time Delay Switch)：

导通时间 t_{on} 和断开时间 t_{off} 的设定必须大于零，而且 t_{on} 不能等于 t_{off}。

若 $t_{on} < t_{off}$，则开关开始时断开，在 t_{on} 时关闭，在 t_{off} 时再次断开。

若 $t_{on} > t_{off}$，则开关开始时关闭，在 t_{off} 时断开，在 t_{on} 时再次关闭。

电压控制开关(Voltage Controlled Switch)：

有导通电压 V_{on} 和断开电压 V_{off}，当控制端的电压大于或等于 V_{on} 时，开关闭合；当控制端的电压小于 V_{off} 时，开关断开。

电流控制开关(Current Controlled Switch)：

当流过控制端的电流大于或等于 I_{on} 时，开关闭合，当流过控制端的电流小于 I_{off} 时，开关断开。

可调电位器(Potentiometer):

$$R=(\text{设定值}/100)*\text{电阻器阻值}$$

可调电容器(Variable Capacitor):

$$C=(\text{设定值}/100)*\text{电容器容量}$$

可调电感器(Variable Inductor):

$$L=(\text{设定值}/100)*\text{电感器容量}$$

无芯线圈(Coreless Coil):

为理想模型。

$$V_{out}=N*I_{in}$$

其中 V_{out}——输出电压;

I_{in}——输入电流。

磁芯(Magnetic Core):

为理想模型。

磁场强度

$$H=mmf/l$$

其中 mmf——磁动力,即输入电压;

l——磁芯长度。

电流

$$I=BA$$

其中 B——磁通量密度;

A——截面积。

全波桥式整流器(Full Wave Rectifier):

平均输出直流电压

$$V_{dc}=0.063*(V_p-1.4)$$

其中 V_p——输入交流电压的峰值幅度。

三端运算放大器(3 TERMINAL Operational Amplifier):

$$V_{out}=AV_{diff}$$

其中 V_{out}——输出电压;

A——开环电压增益;

V_{diff}——输入端的差模电压。

五端运算放大器(5 Terminal Operational Amplifier):

该器件能仿真运算放大器的开环增益、输入/输出阻抗、共模抑制比、失调电压和电流、偏置电流和电压、频率响应、摆率、输出电流和电压极限等参数。

锁相环电路(Phase Locked Loop):

锁相环电路由相位检测电路、低通滤波器和压控振荡器电路组成。

相位检测器的输出

$$V_d=K_d*\sin(\varphi_i-\varphi_o),\qquad \varphi_i=2\pi\int f_i(t)dt$$

低通滤波器的电容器容量

$$C = 1/(2\pi * f_p * R)$$

压控振荡器输出

$$f_o(t) = f_c + K_o * V_c(t), \qquad \varphi_o = 2\pi \int f_o(t)dt$$

其中　f_i——输入频率；

f_p——低通滤波器极点位置；

f_o——VCO 输出频率；

f_c——VCO 自由频率；

V_d——相位检测器输出的直流电压；

V_o——VCO 输出电压；

K_o——VCO 转换增益；

K_d——相位检测器转换增益；

φ_i——输入信号相位；

φ_o——VCO 输出信号相位。

A/D 转换器(Analog to Digital Converter)：

SOC 端为高电平时，开始转换，此时 EOC 被置为低电平，采样时间为 1 μs；当转换过程完成后，ECO 为高电平。在数字输出口 D7～D0 获得对应的二进制数。该电路可以通过使 OE 端为高电平而实现三态输出。

输入信号对应的量化离散电平为

$$V_{in} * 256/V_1$$

满度电压为

$$V_f = V_{ref+} - V_{ref-}$$

输出的二进制电平为

$$BIN[V_{in} * 256/V_{fs}]$$

D/A 转换器(Digital to Analog Converter)(输出电流)：

输出电流为

$$I_o = (I_{ref+} - I_{ref-}) * D/256$$

其中　D——输入二进制数对应的十进制数。

另一个输出端为

$$I_c = K(I_{ref+} - I_{ref-}) - I_o$$

其中，$K = 255/256$。

D/A 转换器(Digital to Analog Converter)(输出电压)：

输出电压

$$V_o = (V_{ref+} - V_{ref-}) * D/256$$

其中　D——输入二进制数对应的十进制数。

单稳态触发器(Conostable Multivibrator)：

为边沿触发的脉冲产生电路，脉冲宽度受 RC 定时电路控制。它有两个输入端，A1 为上升沿触发，A2 为下降沿触发，一旦电路被触发，输入信号将不再起作用。连接方法如下：

① 将电阻 R 和电容 C 串联，电阻 R 和电容 C 的连接点连至器件的 RT/CT 输入端。

② 电容 C 的另一端连至器件的 CT 输入端。

③ 将电阻的另一端连至电源 V_{cc} 端。

电路输出脉宽

$$T_W = 0.069\,3 * R * C$$

触发脉冲的电平可以进行修改和设置。

条形光柱(Barograph Display):

为十个 LED 显示器并排放置,器件的左边是阳极,器件的右边是阴极。当启动电流流过 LED 时,灯会发亮。

带译码条形光柱(Decoded Barograph Display):

点亮每一条灯的电压(从最低段至最高段)为

$$V_{on} = V1 + (Vb - V1) * (n-1)/9$$

其中 n——点亮灯的数量。

电压微分器(Voltage Differentiator):

输出方程

$$V_{out} = K\frac{dV_i}{dt} + V_{off}$$

电压积分器(Voltage Integrator):

输出方程

$$V_{out} = K\int_0^t (V_i(t) + V_{ioff})dt + V_{oic}$$

电压增益模块(Voltage Gain Block):

输出方程

$$V_{out} = K(V_{in} + V_{ioff}) + V_{Ooff}$$

传输函数模块(Transfer Function Block):

传输函数

$$T(s) = Y(s)/X(s) = K * (A_3 s^3 + A_2 s^2 + A_1)/(B_3 s^3 + B_2 s^2 + B_1)$$

乘法器(Multiplier):

输出电压

$$V_{out} = K[X_k(V_x + X_{off}) * Y_k(V_y + Y_{off})] + V_{off}$$

其中 V_x——X 输入电压;

V_y——Y 输入电压;

K——输出增益;

V_{off}——输入失调;

X_k——X 增益;

Y_k——Y 增益;

X_{off}——偏移。

除法器(Divider):

输出电压

$$V_{out} = \left(\frac{(V_x + X_{off}) \times X_k}{(V_y + Y_{off}) \times Y_k}\right) \times K + V_{off}$$

其中　V_x——X 输入电压；

V_y——Y 输入电压；

K——输出增益；

V_{off}——输出失调；

Y_{off}——Y 偏移；

Y_k——Y 增益；

X_{off}——X 偏移；

X_k——X 增益。

三端电压加法器(Three Way Voltage Summer)：

输出电压

$$V_{out}=K_{out}[K_a(V_a+V_{aoff})+K_b(V_b+V_{boff})+K_c(V_c+V_{coff})]+V_{Ooff}$$

电压限幅器(Voltage Limiter)：

器件持性

$$V_{out}=K(V_{in}+V_{ioff}) \qquad V_{min}\leqslant V_{out}\leqslant V_{max}$$

$$V_{out}=V_{max} \qquad V_{out}>V_{max}$$

$$V_{out}=V_{min} \qquad V_{out}<V_{min}$$

电流限幅器模块(Current Limiter Block)：

该器件的沉降源电流极限设置在 10 mA，电路增益为 1，所以输出电流应该是

$$I_{load}=V_{in}*K/R_{load}$$

熔断器(Fuse)特性：

$$R=0, \qquad i_a\leqslant I_{max}$$

$$R=\infty, \qquad i_a>I_{max}$$

数据写入器 (Write Data)：

该器件能以 ASCII 码的形式存储仿真的结果，可以写入时间、各节点电压等有关数据。

数据形式：时间()电压 1()电压 2()……()电压 8。

SPICE 子电路 (SPICE Subcircuit)：

可以将 SPICE 的网表文件作为子电路插入本电路中进行仿真。

有损耗传输线(Lossy Transmission Line)特性方程：

$$\partial v/\partial x=-(L\,\partial i/\partial t+Ri)$$

$$\partial i/\partial x=-(C\,\partial i/\partial t+Gv)$$

边界和初始条件为

$$V(0,t)=V1(t) \qquad V(1,t)=V2(t) \qquad I(0,t)=I1(t)$$

$$I(1,t)=-I2(t) \qquad V(x,0)=V0(X) \qquad I(x,0)=I0(X)$$

其中　L——导线长度；

V(X,t)——在点 X 和时间 t 时的电压；

I(X,t)——在点 X 和时间 t 时的正方向电流；

V(0,t)——在点 0 和时间 t 时的电压；

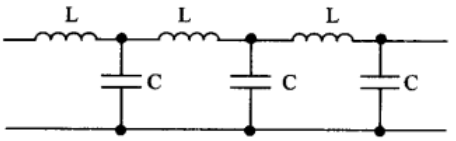
I(0,t)——在点 0 和时间 t 时的正方向电流；

V(X,0)——在点 X 和时间 t 时的正方向电流；

I(X,0)——在点 X 和时间 0 时的正方向电流。

无损耗传输线(Lossless Transmission Line)：

作为理想模型，仿真的是纯电阻性的特性阻抗和传输时间延迟，其值等于 L/C 的均方根值。模型如附录图 1.5 所示。



附录图 1.5 理想模型

$$C_t = T_d/Z \qquad L_t = T_d * Z$$

其中 C_t——单位长度电容量；

L_t——单位长度电感量；

T_d——传输时间延时；

Z——标称阻抗。

传输时间延时为

$$T_d = (L/V_p) \qquad V_p = V_f * C$$

其中 L——导线长度；

V_p——传输速度；

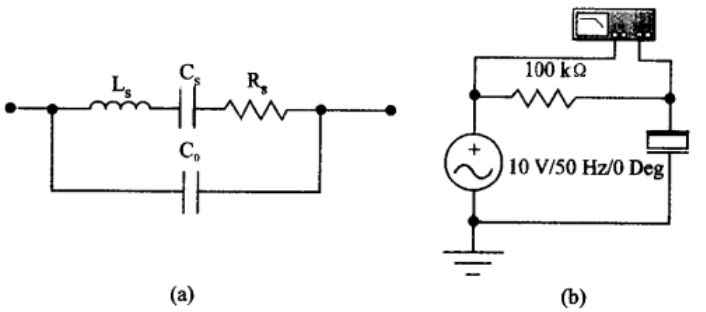
V_f——速度因子；

C——光速。

晶振(Crystal)：

晶体振荡器仿真的等效电路模型如附录图 1.6(a) 所示，晶体谐振特性的测试电路如附录图 1.6(b) 所示。

$$L_s = Q * R_s/(2\pi * f) \qquad Q = 1/(2\pi * f * C_s * R_s) \qquad f = 1/(2\pi * \sqrt{L_s \times C_s})$$



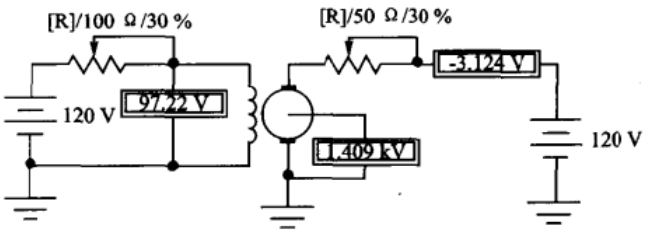
附录图 1.6 等效电路模型

直流电机(DC Motor)：

该部件能仿真理想的直流电机在串联激励和并联激励情况下的特性。在并联方式激励时，将直流电源的正极连至电机的“2”和“4”端，将直流电源的负极连至电机的“1”和“3”端。在串联方式激励时，将电机的“ 2”和“ 3”端相连，再把直流电源的正极连至电机的“4”端，直流电源的负极连至“1”端。在分开激励时，可以将直流电源的正负极分别连至直流电机的“1”和“2”端，把另一个直流电源的正负极分别连至电机的“3”和“4”端。“5”端是电机输出端，输出的是电机的转速值(RPM)。观察电机的输出可以采用以下几个方法：

- 在“5”端和地之间连接一个电压表，当进行仿真时观察电机的输出值；
- 在“5”端用示波器观察输出值，转速值以电压形式表示；
- 在“5”端选择合适的分析方法(比如直流工作点分析)进行观察。

应用举例如附录图 1.7 所示。



附录图 1.7　直流电机应用

直流电机特性方程为

$$V_a = R_a * i_a + L_a * di_a/dt + K_m * i_f * \omega_m$$

$$V_f = R_f * i_f + L * di_f/dt$$

$$J * d\omega_m/dt + B_f * \omega_m + T_L = K_m * i_f * i_a$$

其中　ω_m——旋转速度；

K_m——EMF 常数；

V_a——电枢电压；

V_f——励磁电压。

真空三极管(Triode Vacuum Tube)：

真空三极管的直流特性的仿真采用二维电压控制电流的方法，屏极电流为 I_p。

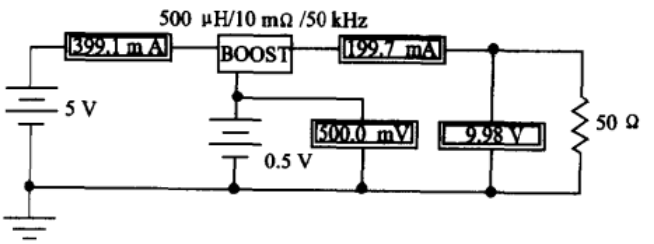
$$I_p = \begin{cases} k(u * V_{gk} + V_{pk})^{3/2} & u * V_{gk} + V_{pk} \geqslant 0 \\ 0 & u * V_{gk} + V_{pk} < 0 \end{cases}$$

开关电源升压转换器(Boost Converter)：

该器件是一种求均电路，用于模拟 DC－DC 开关电源转换器的求均特性。它能模拟电源转换中的小信号和大信号特性，也能模拟 DC－AC 和大信号状态下，电感中电流为连续波和非连续波模式(CCM 和 DCM)下，开关电源的瞬态响应。该器件的一个基本用途是作为电压递增转换器使用。其输出电压 $V_o = V_{in}/(1-D)$。D 是转换电路的开关占空比，取决于控制电压。控制电压越小，占空比也越小，输出电压也越低。

下面的应用举例如附录图 1.8 所示，输入 5 V 电源，输出为 10 V，控制电压为 0.5 V(占空比为 50 %)。调整控制电压为 0.1～0.9 V，可以获得输出电压为 5.5～48 V。电路的转换效率可以用下式计算，即

$$\eta = (V_{out} * I_{out})/(V_{in} * I_{in})$$



附录图 1.8　开关电源升压转换器

器件的特性方程为

$$I_f = I_{LL} + I_{LD} = I_L$$

$$I_o = (I_{LL} + I_{LD}) * D_2/(D_2 + D) = I_L * D_2/(D_2 + D)$$

$$I_{LL} = \frac{1}{L}\int_0^t [D * V_f - D_2(V_o - V_f)]dt$$

其中 D——开关器件的占空比。

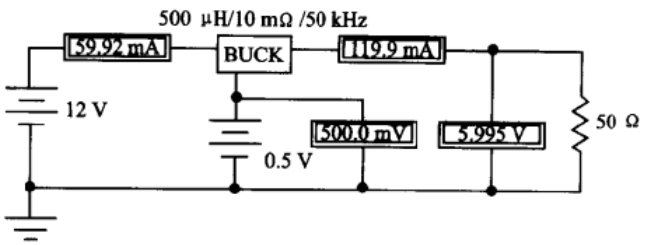
开关电源降压转换器(Buck Converter):

该器件是一种求均电路,用于仿真 DC - DC 开关电源转换器的求均特性。它能模拟电源转换中的小信号和大信号特性,也能模拟 DC - AC 和大信号状态下,电感中电流为连续波和非连续波模式(CCM 和 DCM)下,开关电源的瞬态响应。该器件的一个基本用途是作为递减转换器使用,输出电压 V_o 为

$$V_o = V_{in} * D$$

其中的 D 为转换电路的开关占空比,取决于控制电压,控制电压越小,占空比也越小,输出电压也越低。下面的应用举例如附录图 1.9 所示,输入 12 V 电源,输出为 6 V,调整控制电压 0.1～0.9 V,可以获得输出电压为 1.2～10 V。电路的转换效率可以用下式计算,即

$$\eta = (V_{out} * I_{out})/(V_{in} * I_{in})$$



附录图 1.9 开关电源降压转换器

电压递减转换器特性方程为

$$I_f = (I_{LL} + I_{LD}) * D/(D_2 + D) = I_L * D/(D_2 + D)$$

$$I_o = -(I_{LL} + I_{LD}) = -I_L$$

$$I_{LL} = \frac{1}{L}\int_0^t [D * (V_f - V_o) - D_2 * V_o]dt$$

其中 D——开关器件的占空比。

开关电源升降压转换器(Buck Boost Converter):

该器件是一种求均电路,用于仿真 DC - DC 开关电源转换器的求均特性。它模拟电源转换中的小信号和大信号特性,也能模拟 DC - AC 和大信号状态下,电感中电流为连续和非连续波模式(CCM 和 DCM)下,开关电源的瞬态响应。

$$I_f = (I_{LL} + I_{LD}) * D/(D_2 + D) = I_L * D/(D_2 + D)$$

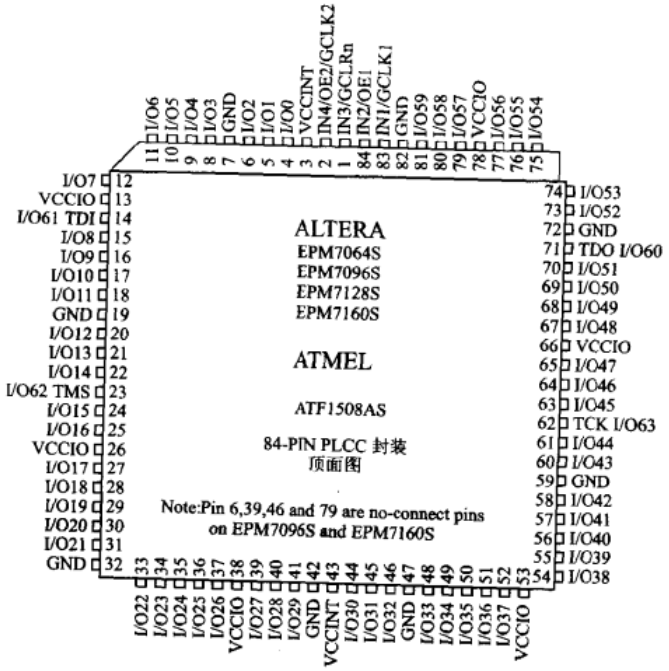
$$I_o = (I_{LL} + I_{LD}) * D/(D_2 + D) = I_L * D/(D_2 + D)$$

$$I_{LL} = \frac{1}{L}\int_0^t [D * V_f - D_2 * V_o]dt$$

其中 D——开关器件的占空比。

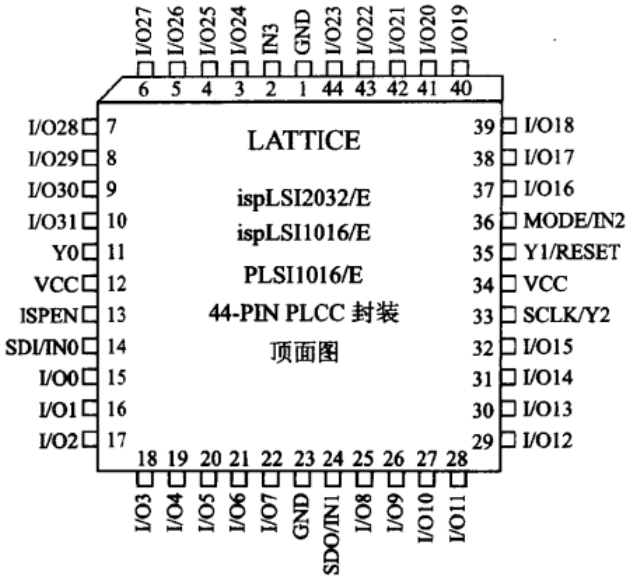
附录二　常用可编程逻辑器件引脚图

常用可编程逻辑器件引脚图如附录图 2.1～附录图 2.5 所示。

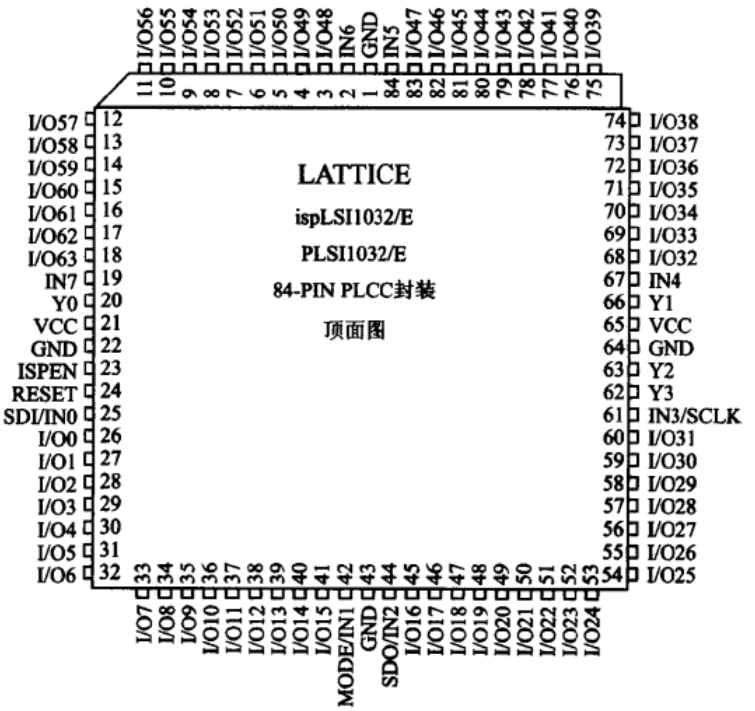


附录图 2.1　MAX7000S 系列 84－PIN 引脚图

附录图 2.2　FLEX10K 系列 84－PIN 引脚图



附录图 2.3 ispLSI2032E/1016E44 - PIN 引脚图



附录图 2.4 ispLSI1032/E84 - PIN 引脚图

附录图 2.5　ispLSI1048/C/E128 - PIN 引脚图